<table>
<thead>
<tr>
<th><strong>Tcl/Tk Applications</strong></th>
<th>The interpreters which implement Tcl and Tk.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl Commands</strong></td>
<td>The commands which the <code>tclsh</code> interpreter implements.</td>
</tr>
<tr>
<td><strong>Tk Commands</strong></td>
<td>The additional commands which the <code>wish</code> interpreter implements.</td>
</tr>
<tr>
<td><strong>Tcl Library</strong></td>
<td>The C functions which a Tcl extended C program may use.</td>
</tr>
<tr>
<td><strong>Tk Library</strong></td>
<td>The additional C functions which a Tk extended C program may use.</td>
</tr>
<tr>
<td><strong>Keywords</strong></td>
<td>The keywords from the Tcl/Tk man pages.</td>
</tr>
</tbody>
</table>
tclsh wish
<table>
<thead>
<tr>
<th>bell</th>
<th>font</th>
<th>options</th>
<th>tk_chooseColor</th>
<th>tk_textPaste</th>
</tr>
</thead>
<tbody>
<tr>
<td>bind</td>
<td>frame</td>
<td>pack</td>
<td>tk_chooseDirectory</td>
<td>tkerror</td>
</tr>
<tr>
<td>bindtags</td>
<td>grab</td>
<td>panedwindow</td>
<td>tk_dialog</td>
<td>tkvars</td>
</tr>
<tr>
<td>bitmap</td>
<td>grid</td>
<td>photo</td>
<td>tk_focusFollowsMouse</td>
<td>tkwait</td>
</tr>
<tr>
<td>button</td>
<td>image</td>
<td>place</td>
<td>tk_focusNext</td>
<td>toplevel</td>
</tr>
<tr>
<td>canvas</td>
<td>keysyms</td>
<td>radiobutton</td>
<td>tk_focusPrev</td>
<td>ttk::button</td>
</tr>
<tr>
<td>checkbutton</td>
<td>label</td>
<td>raise</td>
<td>tk_getOpenFile</td>
<td>ttk::check</td>
</tr>
<tr>
<td>clipboard</td>
<td>labelframe</td>
<td>scale</td>
<td>tk_getSaveFile</td>
<td>ttk::comb</td>
</tr>
<tr>
<td>colors</td>
<td>listbox</td>
<td>scrollbar</td>
<td>tk_menuSetFocus</td>
<td>ttk::entry</td>
</tr>
<tr>
<td>console</td>
<td>loadTk</td>
<td>selection</td>
<td>tk_messageBox</td>
<td>ttk::frame</td>
</tr>
<tr>
<td>cursors</td>
<td>lower</td>
<td>send</td>
<td>tk_optionMenu</td>
<td>ttk::intro</td>
</tr>
<tr>
<td>destroy</td>
<td>menu</td>
<td>spinbox</td>
<td>tk_popup</td>
<td>ttk::label</td>
</tr>
<tr>
<td>entry</td>
<td>menubutton</td>
<td>text</td>
<td>tk_setPalette</td>
<td>ttk::labelf</td>
</tr>
<tr>
<td>event</td>
<td>message</td>
<td>tk</td>
<td>tk_textCopy</td>
<td>ttk::menu</td>
</tr>
<tr>
<td>focus</td>
<td>option</td>
<td>tk_bisque</td>
<td>tk_textCut</td>
<td>ttk::notelb</td>
</tr>
</tbody>
</table>

Copyright © 1990-1994 The Regents of the University of California
Copyright © 1994 The Australian National University
Copyright © 1994-1997 Sun Microsystems, Inc
Copyright © 1995-1997 Roger E. Critchlow Jr
Copyright © 1997-2000 Scrippts Corporation
Copyright © 1998-2000 Ajuba Solutions
Copyright © 2000 Jeffrey Hobbs
Copyright © 2001-2008 Donal K. Fellows
Copyright © 2003 ActiveState Corporation
Copyright © 2004-2006 Joe English
Copyright © 2006-2007 Daniel A. Steffen <das(at)users.sourceforge.net>
Copyright © 2008 Pat Thoyts
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attemptckalloc</td>
<td>Functions to allocate memory</td>
</tr>
<tr>
<td>attemptckrealloc</td>
<td>Functions to reallocate memory</td>
</tr>
<tr>
<td>ckalloc</td>
<td>Functions to allocate memory</td>
</tr>
<tr>
<td>cfree</td>
<td>Functions to free memory</td>
</tr>
<tr>
<td>ckrealloc</td>
<td>Functions to reallocate memory</td>
</tr>
<tr>
<td>Tcl_FSEvalFile</td>
<td>Functions to evaluate file</td>
</tr>
<tr>
<td>Tcl_FSEvalFileEx</td>
<td>Functions to evaluate file</td>
</tr>
<tr>
<td>Tcl_FSFileAttrsGet</td>
<td>Functions to get file attributes</td>
</tr>
<tr>
<td>Tcl_FSFileAttrsSet</td>
<td>Functions to set file attributes</td>
</tr>
<tr>
<td>Tcl_FSFileAttrStrings</td>
<td>Functions to get file attributes</td>
</tr>
<tr>
<td>Tcl_FSFileSystemInfo</td>
<td>Functions to get file system info</td>
</tr>
<tr>
<td>Tcl_FSGetCwd</td>
<td>Functions to get working directory</td>
</tr>
<tr>
<td>Tcl_FSGetFileSystemForPath</td>
<td>Functions to get file system path</td>
</tr>
<tr>
<td>Tcl_FSGetInternalRep</td>
<td>Functions to get internal representation of file system</td>
</tr>
<tr>
<td>Tcl_FSGetNativePath</td>
<td>Functions to get native path</td>
</tr>
<tr>
<td>Tcl_FSGetNormalizedPath</td>
<td>Functions to get normalized path</td>
</tr>
<tr>
<td>Tcl_FSGetPathType</td>
<td>Functions to get path type</td>
</tr>
<tr>
<td>Tcl_FSGetTranslatedPath</td>
<td>Functions to get translated path</td>
</tr>
<tr>
<td>Tcl_FSGetTranslatedStringPath</td>
<td>Functions to get translated string path</td>
</tr>
<tr>
<td>Tcl_FSJoinPath</td>
<td>Functions to join path</td>
</tr>
<tr>
<td>Tcl_FSJoinToPath</td>
<td>Functions to join path</td>
</tr>
<tr>
<td>Tcl_FSLink</td>
<td>Functions to link paths</td>
</tr>
<tr>
<td>Tcl_FSLstat</td>
<td>Functions to list volumes</td>
</tr>
<tr>
<td>Tcl_FSLLoadFile</td>
<td>Functions to load file</td>
</tr>
<tr>
<td>Tcl_FSLstat</td>
<td>Functions to list volumes</td>
</tr>
<tr>
<td>Tcl_FSMatchInDirectory</td>
<td>Functions to match in directory</td>
</tr>
<tr>
<td>Tcl_FSMountsChanged</td>
<td>Functions to mount volumes</td>
</tr>
<tr>
<td>Tcl_FSNewNativePath</td>
<td>Functions to new native path</td>
</tr>
<tr>
<td>Tcl_FSSpeedUpChannel</td>
<td>Functions to speed up file channel</td>
</tr>
<tr>
<td>Tcl_FSPathSeparator</td>
<td>Functions to set path separator</td>
</tr>
<tr>
<td>Tcl_FSRegister</td>
<td>Functions to register</td>
</tr>
<tr>
<td>Tcl_FSRemoveDirectory</td>
<td>Functions to remove directory</td>
</tr>
<tr>
<td>Tcl_FSRenameFile</td>
<td>Functions to rename file</td>
</tr>
</tbody>
</table>

Tcl8.5.8/Tk8.5.8 Documentation > Tcl Library
Tcl_AsyncDelete
Tcl_AsyncInvoke
Tcl_AsyncMark
Tcl_AsyncReady
Tcl_AttemptAlloc
Tcl_AttemptRealloc
Tcl_AttemptSetObjLength
Tcl_BackgroundError
Tcl_Backslash
Tcl_CallWhenDeleted
Tcl_CancelIdleCall
Tcl_ChannelBlockModeProc
Tcl_ChannelBuffered
Tcl_ChannelClose2Proc
Tcl_ChannelCloseProc
Tcl_ChannelFlushProc
Tcl_ChannelGetHandleProc
Tcl_ChannelGetOptionProc
Tcl_ChannelHandlerProc
Tcl_ChannelInputProc
Tcl_ChannelName
Tcl_ChannelOutputProc
Tcl_ChannelSeekProc
Tcl_ChannelSetOptionProc
Tcl_ChannelThreadActionProc
Tcl_ChannelTruncateProc
Tcl_ChannelVersion
Tcl_ChannelWatchProc
Tcl_ChannelWideSeekProc
Tcl_Chtdir
Tcl_ClearChannelHandlers
Tcl_Close

Tcl_FSSplitPath
Tcl_FSStat
Tcl_FSUnregister
Tcl_FSUtime
Tcl_GetAlias
Tcl_GetAliasObj
Tcl_GetAssocData
Tcl_GetBignumFromObj
Tcl_GetBoolean
Tcl_GetBooleanFromObj
Tcl_GetByteArrayFromObj
Tcl_GetChannel
Tcl_GetChannelBufferSize
Tcl_GetChannelError
Tcl_GetChannelErrorInterp
Tcl_GetChannelHandle
Tcl_GetChannelInstanceData
Tcl_GetChannelMode
Tcl_GetChannelName
Tcl_GetChannelNames
Tcl_GetChannelNamesEx
Tcl_GetChannelOption
Tcl_GetChannelThread
Tcl_GetChannelType
Tcl_GetCharLength
Tcl_GetCommandFromObj
Tcl_GetCommandFullName
Tcl_GetCommandInfo
Tcl_GetCommandInfoFromToken
Tcl_GetCommandName
Tcl_GetCurrentNamespace
Tcl_GetCurrentThread
Tcl_GetCwd
Tcl_CommandComplete  Tcl_GetDefaultEncodingDir
Tcl_CommandTraceInfo  Tcl_GetDouble
Tcl_Concat  Tcl_GetDoubleFromObj
Tcl_ConcatObj  Tcl_GetEncoding
Tcl.ConditionFinalize  Tcl_GetEncodingFromObj
Tcl.ConditionNotify  Tcl_GetEncodingName
Tcl.ConditionWait  Tcl_GetEncodingNameFromEnvironment
Tcl_ConvertCountedElement  Tcl_GetEncodingNames
Tcl_ConvertElement  Tcl_GetEncodingSearchPath
Tcl_ConvertToType  Tcl_GetEnsembleFlags
Tcl_CreateAlias  Tcl_GetEnsembleMappingDict
Tcl_CreateAliasObj  Tcl_GetEnsembleNamespace
Tcl_CreateChannel  Tcl_GetEnsembleSubcommandList
Tcl_CreateChannelHandler  Tcl_GetEnsembleUnknownHandler
Tcl_CreateCloseHandler  Tcl_GetErrno
Tcl_CreateCommand  Tcl_GetGlobalNamespace
Tcl_CreateEncoding  Tcl_GetHashKey
Tcl_CreateEnsemble  Tcl_GetHashValue
Tcl_CreateEventSource  Tcl_GetHostName
Tcl_CreateExitHandler  Tcl_GetIndexFromObj
Tcl_CreateFileHandler  Tcl_GetIndexFromObjStruct
Tcl_CreateHashEntry  Tcl_GetInt
Tcl_CreateInterp  Tcl_GetInterpPath
Tcl_CreateMathFunc  Tcl_GetIntFromObj
Tcl_CreateNamespace  Tcl_GetLongFromObj
Tcl_CreateObjCommand  Tcl_GetMaster
Tcl_CreateObjTrace  Tcl_GetMathFuncInfo
Tcl_CreateSlave  Tcl_GetNameOfExecutable
Tcl_CreateThread  Tcl_GetNamespaceUnknownHandler
Tcl_CreateThreadExitHandler  Tcl_GetObjResult
Tcl_CreateTimerHandler  Tcl_GetObjType
Tcl_CreateTrace  Tcl_GetOpenFile
Tcl_CutChannel  Tcl_GetPathType
Tcl_DecrRefCount  
Tcl_DeleteAssocData  
Tcl_DeleteChannelHandler  
Tcl_DeleteCloseHandler  
Tcl_DeleteCommand  
Tcl_DeleteCommandFromToken  
Tcl_DeleteEvents  
Tcl_DeleteEventSource  
Tcl_DeleteExitHandler  
Tcl_DeleteFileHandler  
Tcl_DeleteHashEntry  
Tcl_DeleteHashTable  
Tcl_DeleteInterp  
Tcl_DeleteNamespace  
Tcl_DeleteThreadExitHandler  
Tcl_DeleteTimerHandler  
Tcl_DeleteTrace  
Tcl_DetachChannel  
Tcl_DetachPids  
Tcl_DictObjDone  
Tcl_DictObjFirst  
Tcl_DictObjGet  
Tcl_DictObjNext  
Tcl_DictObjPut  
Tcl_DictObjPutKeyList  
Tcl_DictObjRemove  
Tcl_DictObjRemoveKeyList  
Tcl_DictObjSize  
Tcl_DiscardInterpState  
Tcl_DiscardResult  
Tcl_DontCallWhenDeleted  
Tcl_DoOneEvent  
Tcl_DoWhenIdle  
Tcl_GetRange  
Tcl_GetRegExpFromObj  
Tcl_GetReturnOptions  
Tcl_Gets  
Tcl_GetServiceMode  
Tcl_GetSlave  
Tcl_GetObj  
Tcl_GetStackedChannel  
Tcl_GetStdChannel  
Tcl_GetString  
Tcl_GetStringFromObj  
Tcl_GetStringResult  
Tcl_GetThreadData  
Tcl_GetTime  
Tcl_GetTopChannel  
Tcl_GetUniChar  
Tcl_GetUnicode  
Tcl_GetUnicodeFromObj  
Tcl_GetVar  
Tcl_GetVar2  
Tcl_GetVar2Ex  
Tcl_GetVersion  
Tcl_GetWideIntFromObj  
Tcl_GlobalEval  
Tcl_GlobalEvalObj  
Tcl_HashStats  
Tcl_HideCommand  
Tcl_Import  
Tcl_Init  
Tcl_InitCount  
Tcl_InitCustomHashTable  
Tcl_InitHashTable  
Tcl_InitHashTable  
Tcl_InitMemory
Tcl_DStringAppend
Tcl_DStringAppendElement
Tcl_DStringEndSublist
Tcl_DStringFree
Tcl_DStringGetResult
Tcl_DStringInit
Tcl_DStringLength
Tcl_DStringResult
Tcl_DStringSetLength
Tcl_DStringStartSublist
Tcl_DStringTrunc
Tcl_DStringValue
Tcl_DumpActiveMemory
Tcl_DuplicateObj
Tcl_Eof
Tcl_Errnold
Tcl_ErrnoMsg
Tcl_Eval
Tcl_EvalEx
Tcl_EvalFile
Tcl_EvalObjEx
Tcl_EvalObjv
Tcl_EvalTokens
Tcl_EvalTokensStandard
Tcl_EventuallyFree
Tcl.Exit
Tcl.ExitThread
Tcl_Export
Tcl_ExposeCommand
Tcl_ExprBoolean
Tcl_ExprBooleanObj
Tcl_ExprDouble
Tcl_ExprDoubleObj
Tcl_InitNotifier
Tcl_InitObjHashTable
Tcl_InitObjHashTable
Tcl_InitStubs
Tcl_InputBlocked
Tcl_InputBuffered
Tcl_Interp
Tcl_InterpDeleted
Tcl_InvalidateStringRep
Tcl_IsChannelExisting
Tcl_IsChannelRegistered
Tcl_IsChannelShared
Tcl_Is Ensemble
Tcl_IsSafe
Tcl_IsShared
Tcl_IsStandardChannel
Tcl_JoinPath
Tcl_JoinThread
Tcl_LimitAddHandler
Tcl_LimitCheck
Tcl_LimitExceeded
Tcl_LimitGetCommands
Tcl_LimitGetGranularity
Tcl_LimitGetTime
Tcl_LimitReady
Tcl_LimitRemoveHandler
Tcl_LimitSetCommands
Tcl_LimitSetGranularity
Tcl_LimitSetTime
Tcl_LimitTypeEnabled
Tcl_LimitTypeExceeded
Tcl_LimitTypeReset
Tcl_LimitTypeSet
Tcl_LinkVar
<table>
<thead>
<tr>
<th>Tk Library</th>
<th>Tcl/Tk Applications</th>
<th>Tk Commands</th>
<th>Tcl Library</th>
<th>Tk Commands</th>
<th>Tk Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_3DBorderColor</td>
<td>Tk_DisplayName</td>
<td>Tk_GetRel</td>
<td>Tk_3DBorderColor</td>
<td>Tk_DisplayName</td>
<td>Tk_GetRel</td>
</tr>
<tr>
<td>Tk_3DBorderGC</td>
<td>Tk_DistanceToTextLayout</td>
<td>Tk_GetRoi</td>
<td>Tk_3DBorderFromObj</td>
<td>Tk_DistanceToTextLayout</td>
<td>Tk_GetRoi</td>
</tr>
<tr>
<td>Tk_3DHorizontalBevel</td>
<td>Tk_Draw3DPolygon</td>
<td>Tk_GetScr</td>
<td>Tk_Alloc3DBorderFromObj</td>
<td>Tk_Draw3DPolygon</td>
<td>Tk_GetScr</td>
</tr>
<tr>
<td>Tk_3DVerticalBevel</td>
<td>Tk_Draw3DRectangle</td>
<td>Tk_GetScr</td>
<td>Tk_AllocBitmapFromObj</td>
<td>Tk_Draw3DRectangle</td>
<td>Tk_GetScr</td>
</tr>
<tr>
<td>Tk_AddOption</td>
<td>Tk_DrawChars</td>
<td>Tk_GetScr</td>
<td>Tk_AllocColorFromObj</td>
<td>Tk_DrawChars</td>
<td>Tk_GetScr</td>
</tr>
<tr>
<td>Tk_Alloc3DBorderFromObj</td>
<td>Tk_DrawFocusHighlight</td>
<td>Tk_GetSel</td>
<td>Tk_AllocCursorFromObj</td>
<td>Tk_DrawFocusHighlight</td>
<td>Tk_GetSel</td>
</tr>
<tr>
<td>Tk_AllocBitmapFromObj</td>
<td>Tk_DrawTextLayout</td>
<td>Tk_GetUid</td>
<td>Tk_AllocFontFromObj</td>
<td>Tk_DrawTextLayout</td>
<td>Tk_GetUid</td>
</tr>
<tr>
<td>Tk_AllocColorFromObj</td>
<td>Tk_Fill3DPolygon</td>
<td>Tk_GetUse</td>
<td>Tk_AttachHWND</td>
<td>Tk_Fill3DPolygon</td>
<td>Tk_GetUse</td>
</tr>
<tr>
<td>Tk_AllocCursorFromObj</td>
<td>Tk_Fill3DRectangle</td>
<td>Tk_GetVisi</td>
<td>Tk_Attributes</td>
<td>Tk_Fill3DRectangle</td>
<td>Tk_GetVisi</td>
</tr>
<tr>
<td>Tk_AllocFontFromObj</td>
<td>Tk_FindPhoto</td>
<td>Tk_GetVR</td>
<td>Tk_BindEvent</td>
<td>Tk_FindPhoto</td>
<td>Tk_GetVR</td>
</tr>
<tr>
<td>Tk_AttachHWND</td>
<td>Tk_FontId</td>
<td>Tk_Handle</td>
<td>Tk_CanvasDrawableCoords</td>
<td>Tk_FontId</td>
<td>Tk_Handle</td>
</tr>
<tr>
<td>Tk_Attributes</td>
<td>Tk_Free3DBorder</td>
<td>Tk_HHeight</td>
<td>Tk_CanvasEventuallyRedraw</td>
<td>Tk_Free3DBorder</td>
<td>Tk_HHeight</td>
</tr>
<tr>
<td>Tk_BindEvent</td>
<td>Tk_Free3DBorderFromObj</td>
<td>Tk_IdToWi</td>
<td>Tk_CanvasGetCoord</td>
<td>Tk_Free3DBorderFromObj</td>
<td>Tk_IdToWi</td>
</tr>
<tr>
<td>Tk_CanvasDrawableCoords</td>
<td>Tk_FreeBitmap</td>
<td>Tk_HWND</td>
<td>Tk_CanvasPsBitmap</td>
<td>Tk_FreeBitmap</td>
<td>Tk_HWND</td>
</tr>
<tr>
<td>Tk_CanvasEventuallyRedraw</td>
<td>Tk_FreeBitmapFromObj</td>
<td>Tk_IdToWi</td>
<td>Tk_FreeColorFromObj</td>
<td>Tk_FreeBitmapFromObj</td>
<td>Tk_IdToWi</td>
</tr>
<tr>
<td>Tk_CanvasGetCoord</td>
<td>Tk_FreeColor</td>
<td>Tk_ImageC</td>
<td>Tk_FreeColorFromObj</td>
<td>Tk_FreeColor</td>
<td>Tk_ImageC</td>
</tr>
<tr>
<td>Tk_CanvasPsBitmap</td>
<td>Tk_FreeColorFromObj</td>
<td>Tk_Init</td>
<td>Tk_FreeColormap</td>
<td>Tk_FreeColorFromObj</td>
<td>Tk_Init</td>
</tr>
<tr>
<td>Tk_CanvasPsColor</td>
<td>Tk_FreeColormap</td>
<td>Tk_InitCon</td>
<td>Tk_FreeConfigOptions</td>
<td>Tk_FreeColormap</td>
<td>Tk_InitCon</td>
</tr>
<tr>
<td>Tk_CanvasPsFont</td>
<td>Tk_FreeConfigOptions</td>
<td>Tk_InitImage</td>
<td>Tk_FreeCursor</td>
<td>Tk_FreeConfigOptions</td>
<td>Tk_InitImage</td>
</tr>
<tr>
<td>Tk_CanvasPsPath</td>
<td>Tk_FreeCursor</td>
<td>Tk_InitOpti</td>
<td>Tk_FreeCursorFromObj</td>
<td>Tk_FreeCursor</td>
<td>Tk_InitOpti</td>
</tr>
<tr>
<td>Tk_CanvasPsStipple</td>
<td>Tk_FreeCursorFromObj</td>
<td>Tk_InitStul</td>
<td>Tk_FreeFont</td>
<td>Tk_FreeCursorFromObj</td>
<td>Tk_InitStul</td>
</tr>
<tr>
<td>Tk_CanvasPsY</td>
<td>Tk_FreeFont</td>
<td>Tk_Interna</td>
<td>Tk_FreeFontFromObj</td>
<td>Tk_FreeFont</td>
<td>Tk_Interna</td>
</tr>
<tr>
<td>Tk_CanvasSetStippleOrigin</td>
<td>Tk_FreeFontFromObj</td>
<td>Tk_Interna</td>
<td>Tk_FreeGC</td>
<td>Tk_FreeFontFromObj</td>
<td>Tk_Interna</td>
</tr>
<tr>
<td>Tk_CanvasTagsOption</td>
<td>Tk_FreeGC</td>
<td>Tk_Interna</td>
<td>Tk_FreeImage</td>
<td>Tk_FreeGC</td>
<td>Tk_Interna</td>
</tr>
<tr>
<td>Tk_CanvasTextInfo</td>
<td>Tk_FreeImage</td>
<td>Tk_Interna</td>
<td>Tk_FreeOptions</td>
<td>Tk_FreeImage</td>
<td>Tk_Interna</td>
</tr>
<tr>
<td>Tk_CanvasTkwin</td>
<td>Tk_FreeOptions</td>
<td>Tk_InternaA</td>
<td>Tk_FreePixmap</td>
<td>Tk_FreeOptions</td>
<td>Tk_InternaA</td>
</tr>
<tr>
<td>Tk_CanvasWindowCoords</td>
<td>Tk_FreeSavedOptions</td>
<td>Tk_Interse</td>
<td>Tk_FreeSavedOptions</td>
<td>Tk_FreeSavedOptions</td>
<td>Tk_Interse</td>
</tr>
<tr>
<td>Function Name</td>
<td>Function Name</td>
<td>Function Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>----------------------------------------</td>
<td>----------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_ChangeWindowAttributes</td>
<td>Tk_FreeTextLayout</td>
<td>Tk_IsContainer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CharBbox</td>
<td>Tk_FreeXId</td>
<td>Tk_IsEmbedded</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_Class</td>
<td>Tk_GeometryRequest</td>
<td>Tk_IsMapped</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_ClearSelection</td>
<td>Tk_Get3DBorder</td>
<td>Tk_IsTopLevel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_ClipboardAppend</td>
<td>Tk_Get3DBorderFromObj</td>
<td>Tk_Main</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_ClipboardClear</td>
<td>Tk_GetAllBindings</td>
<td>Tk_MainLoc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CollapseMotionEvents</td>
<td>Tk_GetAnchor</td>
<td>Tk_Maintain</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_Colormap</td>
<td>Tk_GetAnchorFromObj</td>
<td>Tk_MainW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_ComputeTextLayout</td>
<td>Tk_GetAtomName</td>
<td>Tk_MakeW</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_ConfigureInfo</td>
<td>Tk_GetBinding</td>
<td>Tk_Manager</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_ConfigureValue</td>
<td>Tk_GetBitmap</td>
<td>Tk_MapWin</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_ConfigureWidget</td>
<td>Tk_GetBitmapFromObj</td>
<td>Tk_MeasureChars</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_ConfigureWindow</td>
<td>Tk_GetCapStyle</td>
<td>Tk_MinRec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CoordsToWindow</td>
<td>Tk_GetColor</td>
<td>Tk_MinRec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateBinding</td>
<td>Tk_GetColorByValue</td>
<td>Tk_MoveR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateBindingTable</td>
<td>Tk_GetColorFromObj</td>
<td>Tk_MoveT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateClientMessageHandler</td>
<td>Tk_GetColormap</td>
<td>Tk_MoveV</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateErrorHandler</td>
<td>Tk_GetCursor</td>
<td>Tk_Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateEventHandler</td>
<td>Tk_GetCursorFromData</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateGenericHandler</td>
<td>Tk_GetCursorFromObj</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateImageType</td>
<td>Tk_GetDash</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateItemType</td>
<td>Tk_GetFont</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateOptionTable</td>
<td>Tk_GetFontFromObj</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreatePhotoImageFormat</td>
<td>Tk_GetFontMetrics</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateSelHandler</td>
<td>Tk_GetGC</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateWindow</td>
<td>Tk_GetHINSTANCE</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_CreateWindowFromPath</td>
<td>Tk_GetHWND</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DefineBitmap</td>
<td>Tk_GetImage</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DefineCursor</td>
<td>Tk_GetImageMasterData</td>
<td>Tk_NameC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DeleteAllBindings</td>
<td>Tk_GetItemTypes</td>
<td>Tk_NameT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DeleteBinding</td>
<td>Tk_GetJoinStyle</td>
<td>Tk_Offset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DeleteBindingTable</td>
<td>Tk_GetJustify</td>
<td>Tk_OwnSec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DeleteClientMessageHandler</td>
<td>Tk_GetJustifyFromObj</td>
<td>Tk_Parent</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Function</td>
<td>Function</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DeleteErrorHandler</td>
<td>Tk_GetMMFromObj</td>
<td>Tk_ParseA</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DeleteEventHandler</td>
<td>Tk_GetNumMainWindows</td>
<td>Tk_PathNc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DeleteGenericHandler</td>
<td>Tk_GetOption</td>
<td>Tk_PhotoE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DeleteImage</td>
<td>Tk_GetOptionInfo</td>
<td>Tk_PhotoE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DeleteMainWindow</td>
<td>Tk_GetOptionValue</td>
<td>Tk_PhotoE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_DestroyWindow</td>
<td>Tk_GetPixels</td>
<td>Tk_PhotoE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_Display</td>
<td>Tk_GetPixelsFromObj</td>
<td>Tk_PhotoE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_PathName</td>
<td>Tk_GetPixmap</td>
<td>Tk_PhotoE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tk_Depth</td>
<td>Tk_GetRelief</td>
<td>Tk_PhotoS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1989-1994 The Regents of the University of California
Copyright © 1994 The Australian National University
Copyright © 1994-1998 Sun Microsystems, Inc
Copyright © 1995-1997 Roger E. Critchlow Jr
Copyright © 1998-2000 Scriptics Corporation
Copyright © 2000 Ajuba Solutions
Copyright © 2002 ActiveState Corporation
Copyright © 2003-2004 Joe English
Copyright © 2007 ActiveState Software Inc
### Tcl/Tk Applications
The interpreters which implement Tcl and Tk.

### Tcl Commands
The commands which the `tclsh` interpreter implements.

### Tk Commands
The additional commands which the `wish` interpreter implements.

### Tcl Library
The C functions which a Tcl extended C program may use.

### Tk Library
The additional C functions which a Tk extended C program may use.

### Keywords
The keywords from the Tcl/Tk man pages.
NAME
tclsh - Simple shell containing Tcl interpreter

SYNOPSIS

tclsh

DESCRIPTION

Tclsh is a shell-like application that reads Tcl commands from its standard input or from a file and evaluates them. If invoked with no arguments then it runs interactively, reading Tcl commands from standard input and printing command results and error messages to standard output. It runs until the exit command is invoked or until it reaches end-of-file on its standard input. If there exists a file .tclshrc (or tclshrc.tcl on the Windows platforms) in the home directory of the user,
interactive `tclsh` evaluates the file as a Tcl script just before reading the first command from standard input.

**SCRIPT FILES**

If `tclsh` is invoked with arguments then the first few arguments specify the name of a script file, and, optionally, the encoding of the text data stored in that script file. Any additional arguments are made available to the script as variables (see below). Instead of reading commands from standard input `tclsh` will read Tcl commands from the named file; `tclsh` will exit when it reaches the end of the file. The end of the file may be marked either by the physical end of the medium, or by the character, “\032” (“\u001a”, control-Z). If this character is present in the file, the `tclsh` application will read text up to but not including the character. An application that requires this character in the file may safely encode it as “\032”, “\x1a”, or “\u001a”; or may generate it by use of commands such as `format` or `binary`. There is no automatic evaluation of `.tclshrc` when the name of a script file is presented on the `tclsh` command line, but the script file can always `source` it if desired.

If you create a Tcl script in a file whose first line is

```
#!/usr/local/bin/tclsh
```

then you can invoke the script file directly from your shell if you mark the file as executable. This assumes that `tclsh` has been installed in the default location in `/usr/local/bin`; if it is installed somewhere else then you will have to modify the above line to match. Many UNIX systems do not allow the `#!` line to exceed about 30 characters in length, so be sure that the `tclsh` executable can be accessed with a short file name.

An even better approach is to start your script files with the following three lines:

```
#!/bin/sh
```
This approach has three advantages over the approach in the previous paragraph. First, the location of the `tclsh` binary does not have to be hard-wired into the script: it can be anywhere in your shell search path. Second, it gets around the 30-character file name limit in the previous approach. Third, this approach will work even if `tclsh` is itself a shell script (this is done on some systems in order to handle multiple architectures or operating systems: the `tclsh` script selects one of several binaries to run). The three lines cause both `sh` and `tclsh` to process the script, but the `exec` is only executed by `sh`. `sh` processes the script first; it treats the second line as a comment and executes the third line. The `exec` statement cause the shell to stop processing and instead to start up `tclsh` to reprocess the entire script. When `tclsh` starts up, it treats all three lines as comments, since the backslash at the end of the second line causes the third line to be treated as part of the comment on the second line.

You should note that it is also common practice to install `tclsh` with its version number as part of the name. This has the advantage of allowing multiple versions of Tcl to exist on the same system at once, but also the disadvantage of making it harder to write scripts that start up uniformly across different versions of Tcl.

**VARIABLES**

`Tclsh` sets the following Tcl variables:

**argc**
Contains a count of the number of `arg` arguments (0 if none), not including the name of the script file.

**argv**
Contains a Tcl list whose elements are the `arg` arguments, in order, or an empty string if there are no `arg` arguments.
argv0
Contains fileName if it was specified. Otherwise, contains the name by which tclsh was invoked.

tcl_interactive
Contains 1 if tclsh is running interactively (no fileName was specified and standard input is a terminal-like device), 0 otherwise.

PROMPTS
When tclsh is invoked interactively it normally prompts for each command with “% ”. You can change the prompt by setting the variables tcl_prompt1 and tcl_prompt2. If variable tcl_prompt1 exists then it must consist of a Tcl script to output a prompt; instead of outputting a prompt tclsh will evaluate the script in tcl_prompt1. The variable tcl_prompt2 is used in a similar way when a newline is typed but the current command is not yet complete; if tcl_prompt2 is not set then no prompt is output for incomplete commands.

STANDARD CHANNELS
See Tcl_StandardChannels for more explanations.

SEE ALSO
encoding, fconfigure, tclvars

KEYWORDS
argument, interpreter, prompt, script file, shell

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

wish - Simple windowing shell

SYNOPSIS

OPTIONS

-encoding name
-colormap new
-display display
-geometry geometry
-name name
-sync
-use id
-visual visual

--

DESCRIPTION

OPTION PROCESSING

APPLICATION NAME AND CLASS

VARIABLES

argc
argv
argv0
geometry
tcl_interactive

SCRIPT FILES

PROMPTS

KEYWORDS

NAME

wish - Simple windowing shell

SYNOPSIS
wish ?-encoding name? ?fileName arg arg ...?

| OPTIONS |

- **-encoding name**
  Specifies the encoding of the text stored in *fileName*. This option is only recognized prior to the *fileName* argument.

- **-colormap new**
  Specifies that the window should have a new private colormap instead of using the default colormap for the screen.

- **-display display**
  Display (and screen) on which to display window.

- **-geometry geometry**
  Initial geometry to use for window. If this option is specified, its value is stored in the **geometry** global variable of the application's Tcl interpreter.

- **-name name**
  Use *name* as the title to be displayed in the window, and as the name of the interpreter for **send** commands.

- **-sync**
  Execute all X server commands synchronously, so that errors are reported immediately. This will result in much slower execution, but it is useful for debugging.

- **-use id**
  Specifies that the main window for the application is to be embedded in the window whose identifier is *id*, instead of being created as an independent toplevel window. *Id* must be specified in the same way as the value for the **-use** option for toplevel widgets (i.e. it has a form like that returned by the **winfo id** command).
  Note that on some platforms this will only work correctly if *id* refers to a Tk **frame** or **toplevel** that has its **-container** option enabled.
-visual visual

Specifies the visual to use for the window. *Visual* may have any of the forms supported by the Tk_GetVisual procedure.

--

Pass all remaining arguments through to the script's argv variable without interpreting them. This provides a mechanism for passing arguments such as -name to a script instead of having wish interpret them.

### DESCRIPTION

**Wish** is a simple program consisting of the Tcl command language, the Tk toolkit, and a main program that reads commands from standard input or from a file. It creates a main window and then processes Tcl commands. If **wish** is invoked with arguments, then the first few arguments, ?-encoding name? ?fileName? specify the name of a script file, and, optionally, the encoding of the text data stored in that script file. A value for *fileName* is recognized if the appropriate argument does not start with “-“.

If there are no arguments, or the arguments do not specify a *fileName*, then wish reads Tcl commands interactively from standard input. It will continue processing commands until all windows have been deleted or until end-of-file is reached on standard input. If there exists a file “.wishrc” in the home directory of the user, **wish** evaluates the file as a Tcl script just before reading the first command from standard input.

If arguments to **wish** do specify a *fileName*, then *fileName* is treated as the name of a script file. **Wish** will evaluate the script in *fileName* (which presumably creates a user interface), then it will respond to events until all windows have been deleted. Commands will not be read from standard input. There is no automatic evaluation of “.wishrc” when the name of a script file is presented on the wish command line, but the script file can always source it if desired.

Note that on Windows, the wishversion.exe program varies from the tclshversion.exe program in an additional important way: it does not
connect to a standard Windows console and is instead a windowed program. Because of this, it additionally provides access to its own console command.

**OPTION PROCESSING**

Wish automatically processes all of the command-line options described in the OPTIONS summary above. Any other command-line arguments besides these are passed through to the application using the argc and argv variables described later.

**APPLICATION NAME AND CLASS**

The name of the application, which is used for purposes such as send commands, is taken from the -name option, if it is specified; otherwise it is taken from fileName, if it is specified, or from the command name by which wish was invoked. In the last two cases, if the name contains a “/” character, then only the characters after the last slash are used as the application name.

The class of the application, which is used for purposes such as specifying options with a RESOURCE_MANAGER property or .Xdefaults file, is the same as its name except that the first letter is capitalized.

**VARIABLES**

Wish sets the following Tcl variables:

**argc**
Contains a count of the number of arg arguments (0 if none), not including the options described above.

**argv**
Contains a Tcl list whose elements are the arg arguments that follow a -- option or do not match any of the options described in OPTIONS above, in order, or an empty string if there are no such arguments.
argv0
Contains $fileName$ if it was specified. Otherwise, contains the name by which wish was invoked.

geometry
If the -geometry option is specified, wish copies its value into this variable. If the variable still exists after $fileName$ has been evaluated, wish uses the value of the variable in a wm geometry command to set the main window's geometry.

tcl_interactive
Contains 1 if wish is reading commands interactively ($fileName$ was not specified and standard input is a terminal-like device), 0 otherwise.

## SCRIPT FILES

If you create a Tcl script in a file whose first line is

```
#!/usr/local/bin/wish
```

then you can invoke the script file directly from your shell if you mark it as executable. This assumes that wish has been installed in the default location in /usr/local/bin; if it is installed somewhere else then you will have to modify the above line to match. Many UNIX systems do not allow the #! line to exceed about 30 characters in length, so be sure that the wish executable can be accessed with a short file name.

An even better approach is to start your script files with the following three lines:

```
#!/bin/sh
# the next line restarts using wish 
exec wish "$0" "$@"
```
This approach has three advantages over the approach in the previous paragraph. First, the location of the wish binary does not have to be hard-wired into the script: it can be anywhere in your shell search path. Second, it gets around the 30-character file name limit in the previous approach. Third, this approach will work even if wish is itself a shell script (this is done on some systems in order to handle multiple architectures or operating systems: the wish script selects one of several binaries to run). The three lines cause both sh and wish to process the script, but the exec is only executed by sh. sh processes the script first; it treats the second line as a comment and executes the third line. The exec statement cause the shell to stop processing and instead to start up wish to reprocess the entire script. When wish starts up, it treats all three lines as comments, since the backslash at the end of the second line causes the third line to be treated as part of the comment on the second line.

The end of a script file may be marked either by the physical end of the medium, or by the character, “\032” ("\u001a", control-Z). If this character is present in the file, the wish application will read text up to but not including the character. An application that requires this character in the file may encode it as “\032”, “\x1a”, or “\u001a”; or may generate it by use of commands such as format or binary.

**PROMPTS**

When wish is invoked interactively it normally prompts for each command with “% ”. You can change the prompt by setting the variables tcl_prompt1 and tcl_prompt2. If variable tcl_prompt1 exists then it must consist of a Tcl script to output a prompt; instead of outputting a prompt wish will evaluate the script in tcl_prompt1. The variable tcl_prompt2 is used in a similar way when a newline is typed but the current command is not yet complete; if tcl_prompt2 is not set then no prompt is output for incomplete commands.

**KEYWORDS**

shell, toolkit
NAME
after - Execute a command after a time delay

SYNOPSIS

after ms
after ms ?script script script ...?
after cancel id
after cancel script script ...
after idle script ?script script ...?
after info ?id?

DESCRIPTION

This command is used to delay execution of the program or to execute a command in background sometime in the future. It has several forms, depending on the first argument to the command:
**after ms**

Ms must be an integer giving a time in milliseconds. The command sleeps for ms milliseconds and then returns. While the command is sleeping the application does not respond to events.

**after ms ?script script script ...?**

In this form the command returns immediately, but it arranges for a Tcl command to be executed ms milliseconds later as an event handler. The command will be executed exactly once, at the given time. The delayed command is formed by concatenating all the *script* arguments in the same fashion as the *concat* command. The command will be executed at global level (outside the context of any Tcl procedure). If an error occurs while executing the delayed command then the background error will be reported by the command registered with *interp bgerror*. The *after* command returns an identifier that can be used to cancel the delayed command using *after cancel*.

**after cancel id**

Cancels the execution of a delayed command that was previously scheduled. *Id* indicates which command should be canceled; it must have been the return value from a previous *after* command. If the command given by *id* has already been executed then the *after cancel* command has no effect.

**after cancel script script ...**

This command also cancels the execution of a delayed command. The *script* arguments are concatenated together with space separators (just as in the *concat* command). If there is a pending command that matches the string, it is cancelled and will never be executed; if no such command is currently pending then the *after cancel* command has no effect.

**after idle script ?script script ...?**

Concatenates the *script* arguments together with space separators (just as in the *concat* command), and arranges for the resulting script to be evaluated later as an idle callback. The script will be run exactly once, the next time the event loop is entered and there
are no events to process. The command returns an identifier that can be used to cancel the delayed command using `after cancel`. If an error occurs while executing the script then the background error will be reported by the command registered with `interp bgerror`.

`after info ?id?`

This command returns information about existing event handlers. If no `id` argument is supplied, the command returns a list of the identifiers for all existing event handlers created by the `after` command for this interpreter. If `id` is supplied, it specifies an existing handler; `id` must have been the return value from some previous call to `after` and it must not have triggered yet or been cancelled. In this case the command returns a list with two elements. The first element of the list is the script associated with `id`, and the second element is either `idle` or `timer` to indicate what kind of event handler it is.

The `after ms` and `after idle` forms of the command assume that the application is event driven: the delayed commands will not be executed unless the application enters the event loop. In applications that are not normally event-driven, such as `tclsh`, the event loop can be entered with the `vwait` and `update` commands.

**EXAMPLES**

This defines a command to make Tcl do nothing at all for $N$ seconds:

```tcl
proc sleep {N} {
    after [expr {int($N * 1000)}]
}
```

This arranges for the command `wake_up` to be run in eight hours (providing the event loop is active at that time):
The following command can be used to do long-running calculations (as represented here by `::my_calc::one_step`, which is assumed to return a boolean indicating whether another step should be performed) in a step-by-step fashion, though the calculation itself needs to be arranged so it can work step-wise. This technique is extra careful to ensure that the event loop is not starved by the rescheduling of processing steps (arranging for the next step to be done using an already-triggered timer event only when the event queue has been drained) and is useful when you want to ensure that a Tk GUI remains responsive during a slow task.

```tcl
proc doOneStep {} {
    if {::my_calc::one_step} {
        after idle [list after 0 doOneStep]
    }
}
doOneStep
```

SEE ALSO

concat, interp, update, vwait

KEYWORDS

cancel, delay, idle callback, sleep, time

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
error - Generate an error

error message ?info? ?code?

DESCRIPTION

Returns a TCL_ERROR code, which causes command interpretation to be unwound. Message is a string that is returned to the application to indicate what went wrong.

The -errorinfo return option of an interpreter is used to accumulate a stack trace of what was in progress when an error occurred; as nested commands unwind, the Tcl interpreter adds information to the -errorinfo return option. If the info argument is present, it is used to initialize the -errorinfo return options and the first increment of unwind information will not be added by the Tcl interpreter. In other words, the command containing the error command will not appear in the stack trace; in its place will be info. Historically, this feature had been most useful in conjunction with the catch command: if a caught error cannot be handled successfully, info can be used to return a stack trace reflecting the original point of occurrence of the error:

```tcl
catch {...} errMsg
set savedInfo ::errorInfo
...
error $errMsg $savedInfo
```
When working with Tcl 8.5 or later, the following code should be used instead:

```tcl
catch {...} errMsg options
... return -options $options $errMsg
```

If the `code` argument is present, then its value is stored in the `-errorcode` return option. The `-errorcode` return option is intended to hold a machine-readable description of the error in cases where such information is available; see the `return` manual page for information on the proper format for this option's value.

**EXAMPLE**

Generate an error if a basic mathematical operation fails:

```tcl
if {1+2 != 3} {
  error "something is very wrong with addition"
}
```

**SEE ALSO**

`catch`, `return`

**KEYWORDS**

`error`
NAME

lappend - Append list elements onto a variable

SYNOPSIS

lappend varName ?value value value ...

DESCRIPTION

This command treats the variable given by varName as a list and appends each of the value arguments to that list as a separate element, with spaces between elements. If varName does not exist, it is created as a list with elements given by the value arguments. Lappend is similar to append except that the values are appended as list elements rather than raw text. This command provides a relatively efficient way to build up large lists. For example, “lappend a $b” is much more efficient than “set a [concat $a [list $b]]” when $a is long.

EXAMPLE

Using lappend to build up a list of numbers.

```tcl
% set var 1
1
% lappend var 2
1 2
% lappend var 3 4 5
1 2 3 4 5
```
SEE ALSO

list, lindex, linsert, llength, lset, lsort, lrange

KEYWORDS

append, element, list, variable

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 2001 Kevin B. Kenny <kennykb(at)acm.org>. All rights reserved.
NAME

platform - System identification support code and utilities

SYNOPSIS

package require platform ?1.0.4?
platform::generic
platform::identify
platform::patterns identifier

DESCRIPTION

The platform package provides several utility commands useful for the identification of the architecture of a machine running Tcl.

Whilst Tcl provides the tcl_platform array for identifying the current architecture (in particular, the platform and machine elements) this is not always sufficient. This is because (on Unix machines) tcl_platform reflects the values returned by the uname command and these are not standardized across platforms and architectures. In addition, on at least one platform (AIX) the tcl_platform(machine) contains the CPU serial number.

Consequently, individual applications need to manipulate the values in tcl_platform (along with the output of system specific utilities) - which is both inconvenient for developers, and introduces the potential for inconsistencies in identifying architectures and in naming conventions.

The platform package prevents such fragmentation - i.e., it establishes a standard naming convention for architectures running Tcl and makes it more convenient for developers to identify the current architecture a
Tcl program is running on.

**COMMANDS**

**platform::identify**
This command returns an identifier describing the platform the Tcl core is running on. The returned identifier has the general format \textit{OS-CPU}. The \textit{OS} part of the identifier may contain details like kernel version, libc version, etc., and this information may contain dashes as well. The \textit{CPU} part will not contain dashes, making the preceding dash the last dash in the result.

**platform::generic**
This command returns a simplified identifier describing the platform the Tcl core is running on. In contrast to \texttt{platform::identify} it leaves out details like kernel version, libc version, etc. The returned identifier has the general format \textit{OS-CPU}.

**platform::patterns \texttt{identifier}**
This command takes an identifier as returned by \texttt{platform::identify} and returns a list of identifiers describing compatible architectures.

**KEYWORDS**

\texttt{operating system, cpu architecture, platform, architecture}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2006 ActiveState Software Inc
NAME
   auto_execok, auto_import, auto_load, auto_mkindex,
   auto_mkindex_old, auto_qualify, auto_reset, tcl_findLibrary,
   parray, tcl_endOfWord, tcl_startOfNextWord,
   tcl_startOfPreviousWord, tcl_wordBreakAfter,
   tcl_wordBreakBefore - standard library of Tcl procedures

SYNOPSIS
INTRODUCTION
COMMAND PROCEDURES
   auto_execok cmd
   auto_import pattern
   auto_load cmd
   auto_mkindex dir pattern pattern ...
   auto_reset
   auto_qualify command namespace
   tcl_findLibrary basename version patch initScript enVarName
      varName
   parray arrayName
   tcl_endOfWord str start
   tcl_startOfNextWord str start
   tcl_startOfPreviousWord str start
   tcl_wordBreakAfter str start
   tcl_wordBreakBefore str start

VARIABLES
   auto_execs
   auto_index
   auto_noexec
   auto_noload
   auto_path
   env(TCL_LIBRARY)
   env(TCLLIBPATH)
tcl_nonwordchars
tcl_wordchars

SEE ALSO
KEYWORDS

NAME

auto_execok, auto_import, auto_load, auto_mkindex,
auto_mkindex_old, auto_qualify, auto_reset, tcl_findLibrary, parray,
tcl_endOfWord, tcl_startOfNextWord, tcl_startOfPreviousWord,
tcl_wordBreakAfter, tcl_wordBreakBefore - standard library of Tcl
procedures

SYNOPSIS

auto_execok cmd
auto_import pattern
auto_load cmd
auto_mkindex dir pattern pattern ...
auto_mkindex_old dir pattern pattern ...
auto_qualify command namespace
auto_reset
tcl_findLibrary basename version patch initScript enVarName
varName
parray arrayName
tcl_endOfWord str start
tcl_startOfNextWord str start
tcl_startOfPreviousWord str start
tcl_wordBreakAfter str start
tcl_wordBreakBefore str start

INTRODUCTION

Tcl includes a library of Tcl procedures for commonly-needed functions.
The procedures defined in the Tcl library are generic ones suitable for
use by many different applications. The location of the Tcl library is
returned by the info library command. In addition to the Tcl library,
each application will normally have its own library of support procedures
as well; the location of this library is normally given by the value of the $app_library global variable, where app is the name of the application. For example, the location of the Tk library is kept in the variable $tk_library.

To access the procedures in the Tcl library, an application should source the file init.tcl in the library, for example with the Tcl command

```
source [file join [info library] init.tcl]
```

If the library procedure Tcl_Init is invoked from an application's Tcl_AppInit procedure, this happens automatically. The code in init.tcl will define the unknown procedure and arrange for the other procedures to be loaded on-demand using the auto-load mechanism defined below.

### COMMAND PROCEDURES

The following procedures are provided in the Tcl library:

**auto_execok cmd**

Determines whether there is an executable file or shell builtin by the name cmd. If so, it returns a list of arguments to be passed to exec to execute the executable file or shell builtin named by cmd. If not, it returns an empty string. This command examines the directories in the current search path (given by the PATH environment variable) in its search for an executable file named cmd. On Windows platforms, the search is expanded with the same directories and file extensions as used by exec. Auto_execok remembers information about previous searches in an array named auto_execs; this avoids the path search in future calls for the same cmd. The command auto_reset may be used to force auto_execok to forget its cached information.

**auto_import pattern**

Auto_import is invoked during namespace import to see if the
imported commands specified by pattern reside in an autoloaded library. If so, the commands are loaded so that they will be available to the interpreter for creating the import links. If the commands do not reside in an autoloaded library, auto_import does nothing. The pattern matching is performed according to the matching rules of namespace import.

**auto_load cmd**
This command attempts to load the definition for a Tcl command named cmd. To do this, it searches an auto-load path, which is a list of one or more directories. The auto-load path is given by the global variable $auto_path if it exists. If there is no $auto_path variable, then the TCLLIBPATH environment variable is used, if it exists. Otherwise the auto-load path consists of just the Tcl library directory. Within each directory in the auto-load path there must be a file tclIndex that describes one or more commands defined in that directory and a script to evaluate to load each of the commands. The tclIndex file should be generated with the auto_mkindex command. If cmd is found in an index file, then the appropriate script is evaluated to create the command. The auto_load command returns 1 if cmd was successfully created. The command returns 0 if there was no index entry for cmd or if the script did not actually define cmd (e.g. because index information is out of date). If an error occurs while processing the script, then that error is returned. Auto_load only reads the index information once and saves it in the array auto_index; future calls to auto_load check for cmd in the array rather than re-reading the index files. The cached index information may be deleted with the command auto_reset. This will force the next auto_load command to reload the index database from disk.

**auto_mkindex dir pattern pattern ...**
Generates an index suitable for use by auto_load. The command searches dir for all files whose names match any of the pattern arguments (matching is done with the glob command), generates an index of all the Tcl command procedures defined in all the matching files, and stores the index information in a file named
**tclIndex** in *dir*. If no pattern is given a pattern of *.*tcl will be assumed. For example, the command

```
    auto_mkindex foo *.tcl
```

will read all the .tcl files in subdirectory foo and generate a new index file foo/tclIndex.

**Auto_mkindex** parses the Tcl scripts by sourcing them into a slave interpreter and monitoring the proc and namespace commands that are executed. Extensions can use the (undocumented) auto_mkindex-parser package to register other commands that can contribute to the auto_load index. You will have to read through auto.tcl to see how this works.

**Auto_mkindex_old** parses the Tcl scripts in a relatively unsophisticated way: if any line contains the word proc as its first characters then it is assumed to be a procedure definition and the next word of the line is taken as the procedure's name. Procedure definitions that do not appear in this way (e.g. they have spaces before the proc) will not be indexed. If your script contains “dangerous” code, such as global initialization code or procedure names with special characters like $, *, [ or ], you are safer using auto_mkindex_old.

**auto_reset**

Destroys all the information cached by auto_execok and auto_load. This information will be re-read from disk the next time it is needed. Auto_reset also deletes any procedures listed in the auto-load index, so that fresh copies of them will be loaded the next time that they are used.

**auto_qualify command namespace**

Computes a list of fully qualified names for command. This list mirrors the path a standard Tcl interpreter follows for command lookups: first it looks for the command in the current namespace,
and then in the global namespace. Accordingly, if command is relative and namespace is not ::, the list returned has two elements: command scoped by namespace, as if it were a command in the namespace namespace; and command as if it were a command in the global namespace. Otherwise, if either command is absolute (it begins with ::), or namespace is ::, the list contains only command as if it were a command in the global namespace.

Auto_qualify is used by the auto-loading facilities in Tcl, both for producing auto-loading indexes such as pkgIndex.tcl, and for performing the actual auto-loading of functions at runtime.

tcl_findLibrary basename version patch initScript enVarName varName
This is a standard search procedure for use by extensions during their initialization. They call this procedure to look for their script library in several standard directories. The last component of the name of the library directory is normally basenameversion (e.g., tk8.0), but it might be “library” when in the build hierarchies. The initScript file will be sourced into the interpreter once it is found. The directory in which this file is found is stored into the global variable varName. If this variable is already defined (e.g., by C code during application initialization) then no searching is done. Otherwise the search looks in these directories: the directory named by the environment variable enVarName; relative to the Tcl library directory; relative to the executable file in the standard installation bin or bin/arch directory; relative to the executable file in the current build tree; relative to the executable file in a parallel build tree.

parray arrayName
Prints on standard output the names and values of all the elements in the array arrayName. ArrayName must be an array accessible to the caller of parray. It may be either local or global.

tcl_endOfWord str start
Returns the index of the first end-of-word location that occurs after
a starting index \textit{start} in the string \textit{str}. An end-of-word location is defined to be the first non-word character following the first word character after the starting point. Returns -1 if there are no more end-of-word locations after the starting point. See the description of \texttt{tcl\_wordchars} and \texttt{tcl\_nonwordchars} below for more details on how Tcl determines which characters are word characters.

\texttt{tcl\_startOfNextWord} \textit{str \textbf{start}}

Returns the index of the first start-of-word location that occurs after a starting index \textit{start} in the string \textit{str}. A start-of-word location is defined to be the first word character following a non-word character. Returns -1 if there are no more start-of-word locations after the starting point.

\texttt{tcl\_startOfPreviousWord} \textit{str \textbf{start}}

Returns the index of the first start-of-word location that occurs before a starting index \textit{start} in the string \textit{str}. Returns -1 if there are no more start-of-word locations before the starting point.

\texttt{tcl\_wordBreakAfter} \textit{str \textbf{start}}

Returns the index of the first word boundary after the starting index \textit{start} in the string \textit{str}. Returns -1 if there are no more boundaries after the starting point in the given string. The index returned refers to the second character of the pair that comprises a boundary.

\texttt{tcl\_wordBreakBefore} \textit{str \textbf{start}}

Returns the index of the first word boundary before the starting index \textit{start} in the string \textit{str}. Returns -1 if there are no more boundaries before the starting point in the given string. The index returned refers to the second character of the pair that comprises a boundary.

\begin{center}
\textbf{VARIABLES}
\end{center}

The following global variables are defined or used by the procedures in the Tcl library:

\texttt{auto\_execs}
Used by `auto_execok` to record information about whether particular commands exist as executable files.

**auto_index**
Used by `auto_load` to save the index information read from disk.

**auto_noexec**
If set to any value, then `unknown` will not attempt to auto-exec any commands.

**auto_noload**
If set to any value, then `unknown` will not attempt to auto-load any commands.

**auto_path**
If set, then it must contain a valid Tcl list giving directories to search during auto-load operations. This variable is initialized during startup to contain, in order: the directories listed in the TCLLIBPATH environment variable, the directory named by the $tcl_library variable, the parent directory of $tcl_library, the directories listed in the $tcl_pkgPath variable.

**env(TCL_LIBRARY)**
If set, then it specifies the location of the directory containing library scripts (the value of this variable will be assigned to the `tcl_library` variable and therefore returned by the command `info library`). If this variable is not set then a default value is used.

**env(TCLLIBPATH)**
If set, then it must contain a valid Tcl list giving directories to search during auto-load operations. Directories must be specified in Tcl format, using “/” as the path separator, regardless of platform. This variable is only used when initializing the `auto_path` variable.

**tcl_nonwordchars**
This variable contains a regular expression that is used by routines like `tcl_endOfWord` to identify whether a character is part of a word or not. If the pattern matches a character, the character is
considered to be a non-word character. On Windows platforms, spaces, tabs, and newlines are considered non-word characters. Under Unix, everything but numbers, letters and underscores are considered non-word characters.

**tcl_wordchars**
This variable contains a regular expression that is used by routines like `tcl_endOfWord` to identify whether a character is part of a word or not. If the pattern matches a character, the character is considered to be a word character. On Windows platforms, words are comprised of any character that is not a space, tab, or newline. Under Unix, words are comprised of numbers, letters or underscores.

**SEE ALSO**

info, re_syntax

**KEYWORDS**

auto-exec, auto-load, library, unknown, word, whitespace
append - Append to variable

append varName ?value value value ...?

Append all of the value arguments to the current value of variable varName. If varName does not exist, it is given a value equal to the concatenation of all the value arguments. The result of this command is the new value stored in variable varName. This command provides an efficient way to build up long variables incrementally. For example, “append a $b” is much more efficient than “set a $a$b” if $a$ is long.

Building a string of comma-separated numbers piecemeal using a loop.

```
set var 0
for {set i 1} {$i<=10} {incr i} {
    append var "," $i
}
puts $var
# Prints 0,1,2,3,4,5,6,7,8,9,10
```

SEE ALSO

concat, lappend
KEYWORDS

append, variable

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
**NAME**

eval - Evaluate a Tcl script

**SYNOPSIS**

eval arg ?arg ...?

**DESCRIPTION**

Eval takes one or more arguments, which together comprise a Tcl script containing one or more commands. Eval concatenates all its arguments in the same fashion as the concat command, passes the concatenated string to the Tcl interpreter recursively, and returns the result of that evaluation (or any error generated by it). Note that the list command quotes sequences of words in such a way that they are not further expanded by the eval command.

**EXAMPLES**

Often, it is useful to store a fragment of a script in a variable and execute it later on with extra values appended. This technique is used in a number of places throughout the Tcl core (e.g. in fcopy, lsort and trace command callbacks). This example shows how to do this using core Tcl commands:

```tcl
set script {
    puts "logging now"
    lappend $myCurrentLogVar
}
set myCurrentLogVar log1
```
# Set up a switch of logging variable part way through!

```tcl
after 20000 set myCurrentLogVar log2

for {set i 0} {$i<10} {incr i} {
    # Introduce a random delay
    after [expr {int(5000 * rand())}]
    update ;# Check for the asynch log switch
    eval $script $i [clock clicks]
}
```

Note that in the most common case (where the script fragment is actually just a list of words forming a command prefix), it is better to use `{*}$script` when doing this sort of invocation pattern. It is less general than the `eval` command, and hence easier to make robust in practice. The following procedure acts in a way that is analogous to the `lappend` command, except it inserts the argument values at the start of the list in the variable:

```tcl
proc lprepend {varName args} {
    upvar 1 $varName var
    # Ensure that the variable exists and contains a lappend var
    # Now we insert all the arguments in one go
    set var [eval [list linsert $var 0 {*}$args]]
}
```

However, the last line would now normally be written without `eval`, like this:

```tcl
set var [linsert $var 0 {*}$args]
```

**SEE ALSO**
catch, concat, error, interp, list, namespace, subst, tclvars, uplevel

KEYWORDS

concatenate, evaluate, script

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

lassign - Assign list elements to variables

SYNOPSIS

lassign list varName ...?

DESCRIPTION

This command treats the value list as a list and assigns successive elements from that list to the variables given by the varName arguments in order. If there are more variable names than list elements, the remaining variables are set to the empty string. If there are more list elements than variables, a list of unassigned elements is returned.

EXAMPLES

An illustration of how multiple assignment works, and what happens when there are either too few or too many elements.

```tcl
lassign {a b c} x y z ;# Empty return
puts $x ;# Prints "a"
puts $y ;# Prints "b"
puts $z ;# Prints "c"

lassign {d e} x y z ;# Empty return
puts $x ;# Prints "d"
puts $y ;# Prints "e"
puts $z ;# Prints ""
```
lassign \{f g h i\} x y  ;# Returns "h i"
puts $x               ;# Prints "f"
puts $y               ;# Prints "g"

The `lassign` command has other uses. It can be used to create the analogue of the “shift” command in many shell languages like this:

```
set ::argv [lassign ::argv argumentToReadOff]
```

See Also

`lindex`, `list`, `lset`, `set`

Keywords

assign, `element`, `list`, `multiple`, `set`, `variable`
NAME

platform::shell - System identification support code and utilities

SYNOPSIS

package require platform::shell ?1.1.4?
platform::shell::generic shell
platform::shell::identify shell
platform::shell::platform shell

DESCRIPTION

The platform::shell package provides several utility commands useful for the identification of the architecture of a specific Tcl shell.

This package allows the identification of the architecture of a specific Tcl shell different from the shell running the package. The only requirement is that the other shell (identified by its path), is actually executable on the current machine.

While for most platform this means that the architecture of the interrogated shell is identical to the architecture of the running shell this is not generally true. A counter example are all platforms which have 32 and 64 bit variants and where a 64bit system is able to run 32bit code. For these running and interrogated shell may have different 32/64 bit settings and thus different identifiers.

For applications like a code repository it is important to identify the architecture of the shell which will actually run the installed packages, versus the architecture of the shell running the repository software.
COMMANDS

platform::shell::identify shell
   This command does the same identification as platform::identify, for the specified Tcl shell, in contrast to the running shell.

platform::shell::generic shell
   This command does the same identification as platform::generic, for the specified Tcl shell, in contrast to the running shell.

platform::shell::platform shell
   This command returns the contents of tcl_platform(platform) for the specified Tcl shell.

KEYWORDS

operating system, cpu architecture, platform, architecture

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2006-2008 ActiveState Software Inc
apply - Apply an anonymous function

apply func ?arg1 arg2 ...?

The command **apply** applies the function *func* to the arguments *arg1* *arg2* ... and returns the result.

The function *func* is a two element list `{args body}` or a three element list `{args body namespace}` (as if the **list** command had been used). The specification of the formal arguments *args* is shared with the **proc** command, and is described in detail in the corresponding manual page.

The contents of *body* are executed by the Tcl interpreter after the local variables corresponding to the formal arguments are given the values of the actual parameters *arg1 arg2* .... When *body* is being executed, variable names normally refer to local variables, which are created automatically when referenced and deleted when **apply** returns. One local variable is automatically created for each of the function's arguments. Global variables can only be accessed by invoking the **global** command or the **upvar** command. Namespace variables can only be accessed by invoking the **variable** command or the **upvar** command.

The invocation of **apply** adds a call frame to Tcl's evaluation stack (the stack of frames accessed via **uplevel**). The execution of *body* proceeds
in this call frame, in the namespace given by *namespace* or in the
global namespace if none was specified. If given, *namespace* is
interpreted relative to the global namespace even if its name does not
start with "::".

The semantics of **apply** can also be described by:

```tcl
proc apply {fun args} {
    set len [llength $fun]
    if {($len < 2) || ($len > 3)} {
        error "can't interpret "$fun" as anonymous f
    }
    lassign $fun argList body ns
    set name ::$ns::[getGloballyUniqueName]
    set body0 {
        rename [lindex [info level 0] 0] {}
    }
    proc $name $argList ${body0}$body
    set code [catch {uplevel 1 $name $args} res opt]
    return -options $opt $res
}
```

**EXAMPLES**

This shows how to make a simple general command that applies a
transformation to each element of a list.

```tcl
proc map {lambda list} {
    set result {}
    foreach item $list {
        lappend result [apply $lambda $item]
    }
    return $result
}
```
map {x {return [string length $x]:$x}} {a bb ccc dddd}
→ 1:a 2:bb 3:ccc 4:dddd
map {x {expr {$x**2 + 3*$x - 2}}}} {-4 -3 -2 -1 0 1 2}
→ 2 -2 -4 -4 -2 2 8 16 26

The **apply** command is also useful for defining callbacks for use in the **trace** command:

```tcl
set vbl "123abc"
trace add variable vbl write {apply {{v1 v2 op} {
    upvar 1 $v1 v
    puts "updated variable to "$v"
}}}
set vbl 123
set vbl abc
```

**SEE ALSO**

**proc**, **uplevel**

**KEYWORDS**

argument, procedure, anonymous function
NAME
  exec - Invoke subprocesses
SYNOPSIS
DESCRIPTION
  -ignorestderr
  -keepnewline
  --
  | |
  |&
  < fileName
  <@ fileId
  << value
  > fileName
  2> fileName
  >& fileName
  >> fileName
  2>> fileName
  >>& fileName
  >@ fileId
  2>@ fileId
  2>@1
  >&@ fileId
PORTABILITY ISSUES
  Windows (all versions)
  Windows NT
  Windows 9x
  Unix
UNIX EXAMPLES
WINDOWS EXAMPLES
SEE ALSO
KEYWORDS
NAME

exec - Invoke subprocesses

SYNOPSIS

exec ?switches? arg ?arg ...?

DESCRIPTION

This command treats its arguments as the specification of one or more subprocesses to execute. The arguments take the form of a standard shell pipeline where each arg becomes one word of a command, and each distinct command becomes a subprocess.

If the initial arguments to exec start with - then they are treated as command-line switches and are not part of the pipeline specification. The following switches are currently supported:

-ignorestderr
    Stops the exec command from treating the output of messages to the pipeline's standard error channel as an error case.

-keepnewline
    Retains a trailing newline in the pipeline's output. Normally a trailing newline will be deleted.

--
    Marks the end of switches. The argument following this one will be treated as the first arg even if it starts with a -.

If an arg (or pair of args) has one of the forms described below then it is used by exec to control the flow of input and output among the subprocess(es). Such arguments will not be passed to the subprocess(es). In forms such as “< fileName”, fileName may either be in a separate argument from “<” or in the same argument with no intervening space (i.e. “<fileName”).
Separates distinct commands in the pipeline. The standard output of the preceding command will be piped into the standard input of the next command.

|&
Separates distinct commands in the pipeline. Both standard output and standard error of the preceding command will be piped into the standard input of the next command. This form of redirection overrides forms such as 2> and >&.

< fileName
The file named by fileName is opened and used as the standard input for the first command in the pipeline.

<@ fileId
FileId must be the identifier for an open file, such as the return value from a previous call to open. It is used as the standard input for the first command in the pipeline. FileId must have been opened for reading.

<< value
Value is passed to the first command as its standard input.

> fileName
Standard output from the last command is redirected to the file named fileName, overwriting its previous contents.

2> fileName
Standard error from all commands in the pipeline is redirected to the file named fileName, overwriting its previous contents.

>& fileName
Both standard output from the last command and standard error from all commands are redirected to the file named fileName, overwriting its previous contents.

>> fileName
Standard output from the last command is redirected to the file
named `fileName`, appending to it rather than overwriting it.

```bash
2>> fileName
```
Standard error from all commands in the pipeline is redirected to the file named `fileName`, appending to it rather than overwriting it.

```bash
>>& fileName
```
Both standard output from the last command and standard error from all commands are redirected to the file named `fileName`, appending to it rather than overwriting it.

```bash
>@ fileId
```
`FileId` must be the identifier for an open file, such as the return value from a previous call to `open`. Standard output from the last command is redirected to `fileId`'s file, which must have been opened for writing.

```bash
2>@ fileId
```
`FileId` must be the identifier for an open file, such as the return value from a previous call to `open`. Standard error from all commands in the pipeline is redirected to `fileId`'s file. The file must have been opened for writing.

```bash
2>@1
```
Standard error from all commands in the pipeline is redirected to the command result. This operator is only valid at the end of the command pipeline.

```bash
>&@ fileId
```
`FileId` must be the identifier for an open file, such as the return value from a previous call to `open`. Both standard output from the last command and standard error from all commands are redirected to `fileId`'s file. The file must have been opened for writing.

If standard output has not been redirected then the `exec` command returns the standard output from the last command in the pipeline, unless "2>@1" was specified, in which case standard error is included as well. If any of the commands in the pipeline exit abnormally or are
killed or suspended, then `exec` will return an error and the error message will include the pipeline's output followed by error messages describing the abnormal terminations; the `-errorcode` return option will contain additional information about the last abnormal termination encountered. If any of the commands writes to its standard error file and that standard error is not redirected and `-ignorestderr` is not specified, then `exec` will return an error; the error message will include the pipeline's standard output, followed by messages about abnormal terminations (if any), followed by the standard error output.

If the last character of the result or error message is a newline then that character is normally deleted from the result or error message. This is consistent with other Tcl return values, which do not normally end with newlines. However, if `-keepnewline` is specified then the trailing newline is retained.

If standard input is not redirected with “<”, “<<” or “<@” then the standard input for the first command in the pipeline is taken from the application's current standard input.

If the last arg is “&” then the pipeline will be executed in background. In this case the `exec` command will return a list whose elements are the process identifiers for all of the subprocesses in the pipeline. The standard output from the last command in the pipeline will go to the application's standard output if it has not been redirected, and error output from all of the commands in the pipeline will go to the application's standard error file unless redirected.

The first word in each command is taken as the command name; tilde-substitution is performed on it, and if the result contains no slashes then the directories in the PATH environment variable are searched for an executable by the given name. If the name contains a slash then it must refer to an executable reachable from the current directory. No “glob” expansion or other shell-like substitutions are performed on the arguments to commands.

**PORTABILITY ISSUES**
Windows (all versions)
Reading from or writing to a socket, using the “@ fileId” notation, does not work. When reading from a socket, a 16-bit DOS application will hang and a 32-bit application will return immediately with end-of-file. When either type of application writes to a socket, the information is instead sent to the console, if one is present, or is discarded.

The Tk console text widget does not provide real standard IO capabilities. Under Tk, when redirecting from standard input, all applications will see an immediate end-of-file; information redirected to standard output or standard error will be discarded.

Either forward or backward slashes are accepted as path separators for arguments to Tcl commands. When executing an application, the path name specified for the application may also contain forward or backward slashes as path separators. Bear in mind, however, that most Windows applications accept arguments with forward slashes only as option delimiters and backslashes only in paths. Any arguments to an application that specify a path name with forward slashes will not automatically be converted to use the backslash character. If an argument contains forward slashes as the path separator, it may or may not be recognized as a path name, depending on the program.

Additionally, when calling a 16-bit DOS or Windows 3.X application, all path names must use the short, cryptic, path format (e.g., using “applba~1.def” instead of “applbakery.default”), which can be obtained with the “file attributes fileName -shortname” command.

Two or more forward or backward slashes in a row in a path refer to a network path. For example, a simple concatenation of the root directory c: with a subdirectory /windows/system will yield c:/windows/system (two slashes together), which refers to the mount point called system on the machine called windows (and the c: is ignored), and is not equivalent to c:/windows/system, which describes a directory on the current computer. The file join command should be used to concatenate path components.
Note that there are two general types of Win32 console applications:


[2] TUI — Textmode User Interface, any application that accesses the console API for doing such things as cursor movement, setting text color, detecting key presses and mouse movement, etc. An example would be telnet.exe from Windows 2000. These types of applications are not common in a Windows environment, but do exist.

exec will not work well with TUI applications when a console is not present, as is done when launching applications under wish. It is desirable to have console applications hidden and detached. This is a designed-in limitation as exec wants to communicate over pipes. The Expect extension addresses this issue when communicating with a TUI application.

Windows NT
When attempting to execute an application, exec first searches for the name as it was specified. Then, in order, .com, .exe, and .bat are appended to the end of the specified name and it searches for the longer name. If a directory name was not specified as part of the application name, the following directories are automatically searched in order when attempting to locate the application:

- The directory from which the Tcl executable was loaded.
- The current directory.
- The Windows NT 32-bit system directory.
- The Windows NT 16-bit system directory.
- The Windows NT home directory.
In order to execute shell built-in commands like `dir` and `copy`, the caller must prepend the desired command with "`cmd.exe /c`" because built-in commands are not implemented using executables.

**Windows 9x**

When attempting to execute an application, `exec` first searches for the name as it was specified. Then, in order, `.com`, `.exe`, and `.bat` are appended to the end of the specified name and it searches for the longer name. If a directory name was not specified as part of the application name, the following directories are automatically searched in order when attempting to locate the application:

- The directory from which the Tcl executable was loaded.
- The current directory.
- The Windows 9x system directory.
- The Windows 9x home directory.
- The directories listed in the path.

In order to execute shell built-in commands like `dir` and `copy`, the caller must prepend the desired command with "`command.com /c`" because built-in commands are not implemented using executables.

Once a 16-bit DOS application has read standard input from a console and then quit, all subsequently run 16-bit DOS applications will see the standard input as already closed. 32-bit applications do not have this problem and will run correctly, even after a 16-bit DOS application thinks that standard input is closed. There is no known workaround for this bug at this time.

Redirection between the **NUL**: device and a 16-bit application does
not always work. When redirecting from **NUL:**, some applications
may hang, others will get an infinite stream of “0x01” bytes, and
some will actually correctly get an immediate end-of-file; the
behavior seems to depend upon something compiled into the
application itself. When redirecting greater than 4K or so to **NUL:**,
some applications will hang. The above problems do not happen
with 32-bit applications.

All DOS 16-bit applications are run synchronously. All standard
input from a pipe to a 16-bit DOS application is collected into a
temporary file; the other end of the pipe must be closed before the
16-bit DOS application begins executing. All standard output or
error from a 16-bit DOS application to a pipe is collected into
temporary files; the application must terminate before the
temporary files are redirected to the next stage of the pipeline. This
is due to a workaround for a Windows 95 bug in the implementation
of pipes, and is how the standard Windows 95 DOS shell handles
pipes itself.

Certain applications, such as **command.com**, should not be
executed interactively. Applications which directly access the
console window, rather than reading from their standard input and
writing to their standard output may fail, hang Tcl, or even hang the
system if their own private console window is not available to them.

**Unix**

The **exec** command is fully functional and works as described.

---

**UNIX EXAMPLES**

Here are some examples of the use of the **exec** command on Unix.

To execute a simple program and get its result:

```
exec uname -a
```

To execute a program that can return a non-zero result, you should
wrap the call to `exec` in `catch` and check the contents of the `-errorcode`
return option if you have an error:

```tcl
set status 0
if {{catch {exec grep foo bar.txt} results options]}}
set details [dict get $options -errorcode]
if {[lindex $details 0] eq "CHILDSTATUS"} {
    set status [lindex $details 2]
} else {
    # Some kind of unexpected failure
}
```

When translating a command from a Unix shell invocation, care should
be taken over the fact that single quote characters have no special
significance to Tcl. Thus:

```bash
awk '{sum += $1} END {print sum}' numbers.list
```

would be translated into something like:

```tcl
exec awk {{sum += $1} END {print sum}} numbers.list
```

If you are converting invocations involving shell globbing, you should
remember that Tcl does not handle globbing or expand things into
multiple arguments by default. Instead you should write things like this:

```tcl
exec ls -l {*}[glob *.tcl]
```

WINDOWS EXAMPLES
Here are some examples of the use of the `exec` command on Windows.

To start an instance of `notepad` editing a file without waiting for the user to finish editing the file:

```
exec notepad myfile.txt &
```

To print a text file using `notepad`:

```
exec notepad /p myfile.txt
```

If a program calls other programs, such as is common with compilers, then you may need to resort to batch files to hide the console windows that sometimes pop up:

```
exec cmp.bat somefile.c -o somefile
```

With the file `cmp.bat` looking something like:

```
@gcc %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Sometimes you need to be careful, as different programs may have the same name and be in the path. It can then happen that typing a command at the DOS prompt finds a different program than the same command run via `exec`. This is because of the (documented) differences in behaviour between `exec` and DOS batch files.

When in doubt, use the command `auto_execok`: it will return the complete path to the program as seen by the `exec` command. This applies especially when you want to run “internal” commands like `dir` from a Tcl script (if you just want to list filenames, use the `glob` command.) To do that, use this:
exec {*[auto_execok dir] *.tcl

SEE ALSO
error, open

KEYWORDS
execute, pipeline, redirection, subprocess

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 2006 Donal K. Fellows.
NAME

lindex - Retrieve an element from a list

SYNOPSIS

lindex list ?index...?

DESCRIPTION

The **lindex** command accepts a parameter, *list*, which it treats as a Tcl list. It also accepts zero or more *indices* into the list. The indices may be presented either consecutively on the command line, or grouped in a Tcl list and presented as a single argument.

If no indices are presented, the command takes the form:

```
    lindex list
```

or

```
    lindex list {}
```

In this case, the return value of **lindex** is simply the value of the *list* parameter.

When presented with a single index, the **lindex** command treats *list* as a Tcl list and returns the *index*’th element from it (0 refers to the first element of the list). In extracting the element, **lindex** observes the same
rules concerning braces and quotes and backslashes as the Tcl command interpreter; however, variable substitution and command substitution do not occur. If index is negative or greater than or equal to the number of elements in value, then an empty string is returned. The interpretation of each simple index value is the same as for the command string index, supporting simple index arithmetic and indices relative to the end of the list.

If additional index arguments are supplied, then each argument is used in turn to select an element from the previous indexing operation, allowing the script to select elements from sublists. The command,

```
lindex $a 1 2 3
```

or

```
lindex $a {1 2 3}
```

is synonymous with

```
lindex [lindex [lindex $a 1] 2] 3
```

**EXAMPLES**

```
lindex {a b c}
  → a b c
lindex {a b c} {}
  → a b c
lindex {a b c} 0
  → a
lindex {a b c} 2
```

\begin{verbatim}
\rightarrow c
lindex {a b c} end
\rightarrow c
lindex {a b c} end-1
\rightarrow b
lindex {{a b c} {d e f} {g h i}} 2 1
\rightarrow h
lindex {{a b c} {d e f} {g h i}} {2 1}
\rightarrow h
lindex {{ {a b} {c d}} { {e f} {g h}}} 1 1 0
\rightarrow g
lindex {{ {a b} {c d}} { {e f} {g h}}} {1 1 0}
\rightarrow g
\end{verbatim}

SEE ALSO

list, lappend, linsert, llength, lsearch, lset, lsort, lrange, lreplace, string

KEYWORDS

element, index, list

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 2001 by Kevin B. Kenny <kennykb(at)acm.org>. All rights reserved.
**proc** - Create a Tcl procedure

**SYNOPSIS**

```tcl
proc name args body
```

**DESCRIPTION**

The `proc` command creates a new Tcl procedure named `name`, replacing any existing command or procedure there may have been by that name. Whenever the new command is invoked, the contents of `body` will be executed by the Tcl interpreter. Normally, `name` is unqualified (does not include the names of any containing namespaces), and the new procedure is created in the current namespace. If `name` includes any namespace qualifiers, the procedure is created in the specified namespace. `Args` specifies the formal arguments to the procedure. It consists of a list, possibly empty, each of whose elements specifies one argument. Each argument specifier is also a list with either one or two fields. If there is only a single field in the specifier then it is the name of the argument; if there are two fields, then the first is the argument name and the second is its default value. Arguments with default values that are followed by non-defaulted arguments become required arguments. In 8.6 this will be considered an error.

When `name` is invoked a local variable will be created for each of the formal arguments to the procedure; its value will be the value of corresponding argument in the invoking command or the argument's default value. Actual arguments are assigned to formal arguments strictly in order. Arguments with default values need not be specified in
a procedure invocation. However, there must be enough actual arguments for all the formal arguments that do not have defaults, and there must not be any extra actual arguments. Arguments with default values that are followed by non-defaulted arguments become required arguments (in 8.6 it will be considered an error). There is one special case to permit procedures with variable numbers of arguments. If the last formal argument has the name `args`, then a call to the procedure may contain more actual arguments than the procedure has formals. In this case, all of the actual arguments starting at the one that would be assigned to `args` are combined into a list (as if the `list` command had been used); this combined value is assigned to the local variable `args`.

When `body` is being executed, variable names normally refer to local variables, which are created automatically when referenced and deleted when the procedure returns. One local variable is automatically created for each of the procedure's arguments. Other variables can only be accessed by invoking one of the `global`, `variable`, `upvar` or `namespace upvar` commands.

The `proc` command returns an empty string. When a procedure is invoked, the procedure's return value is the value specified in a `return` command. If the procedure does not execute an explicit `return`, then its return value is the value of the last command executed in the procedure's body. If an error occurs while executing the procedure body, then the procedure-as-a-whole will return that same error.

**EXAMPLES**

This is a procedure that accepts arbitrarily many arguments and prints them out, one by one.

```bash
proc printArguments args {
    foreach arg $args {
        puts $arg
    }
}
```
This procedure is a bit like the `incr` command, except it multiplies the contents of the named variable by the value, which defaults to 2:

```tcl
proc mult {varName {multiplier 2}} {
    upvar 1 $varName var
    set var [expr {$var * $multiplier}]
}
```

SEE ALSO

`info`, `unknown`

KEYWORDS

`argument`, `procedure`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
array - Manipulate array variables

SYNOPSIS

DESCRIPTION
array anymore arrayName searchId
array donesearch arrayName searchId
array exists arrayName
array get arrayName ?pattern?
array names arrayName ?mode? ?pattern?
array nextelement arrayName searchId
array set arrayName list
array size arrayName
array startsearch arrayName
array statistics arrayName
array unset arrayName ?pattern?

EXAMPLES

SEE ALSO

KEYWORDS

NAME
array - Manipulate array variables

SYNOPSIS

array option arrayName ?arg arg ...?

DESCRIPTION

This command performs one of several operations on the variable given by arrayName. Unless otherwise specified for individual commands below, arrayName must be the name of an existing array variable. The
option argument determines what action is carried out by the command. The legal options (which may be abbreviated) are:

**array anymore arrayName searchId**
Returns 1 if there are any more elements left to be processed in an array search, 0 if all elements have already been returned. *SearchId* indicates which search on *arrayName* to check, and must have been the return value from a previous invocation of **array startsearch**. This option is particularly useful if an array has an element with an empty name, since the return value from **array nextelement** will not indicate whether the search has been completed.

**array donesearch arrayName searchId**
This command terminates an array search and destroys all the state associated with that search. *SearchId* indicates which search on *arrayName* to destroy, and must have been the return value from a previous invocation of **array startsearch**. Returns an empty string.

**array exists arrayName**
Returns 1 if *arrayName* is an array variable, 0 if there is no variable by that name or if it is a scalar variable.

**array get arrayName ?pattern?**
Returns a list containing pairs of elements. The first element in each pair is the name of an element in *arrayName* and the second element of each pair is the value of the array element. The order of the pairs is undefined. If *pattern* is not specified, then all of the elements of the array are included in the result. If *pattern* is specified, then only those elements whose names match *pattern* (using the matching rules of **string match**) are included. If *arrayName* is not the name of an array variable, or if the array contains no elements, then an empty list is returned. If traces on the array modify the list of elements, the elements returned are those that exist both before and after the call to **array get**.

**array names arrayName ?mode? ?pattern?**
Returns a list containing the names of all of the elements in the array that match pattern. Mode may be one of -exact, -glob, or -regexp. If specified, mode designates which matching rules to use to match pattern against the names of the elements in the array. If not specified, mode defaults to -glob. See the documentation for string match for information on glob style matching, and the documentation for regexp for information on regexp matching. If pattern is omitted then the command returns all of the element names in the array. If there are no (matching) elements in the array, or if arrayName is not the name of an array variable, then an empty string is returned.

array nextelement arrayName searchId
Returns the name of the next element in arrayName, or an empty string if all elements of arrayName have already been returned in this search. The searchId argument identifies the search, and must have been the return value of an array startsearch command. Warning: if elements are added to or deleted from the array, then all searches are automatically terminated just as if array donesearch had been invoked; this will cause array nextelement operations to fail for those searches.

array set arrayName list
Sets the values of one or more elements in arrayName. list must have a form like that returned by array get, consisting of an even number of elements. Each odd-numbered element in list is treated as an element name within arrayName, and the following element in list is used as a new value for that array element. If the variable arrayName does not already exist and list is empty, arrayName is created with an empty array value.

array size arrayName
Returns a decimal string giving the number of elements in the array. If arrayName is not the name of an array then 0 is returned.

array startsearch arrayName
This command initializes an element-by-element search through the array given by arrayName, such that invocations of the array
nextelement command will return the names of the individual elements in the array. When the search has been completed, the array donesearch command should be invoked. The return value is a search identifier that must be used in array nextelement and array donesearch commands; it allows multiple searches to be underway simultaneously for the same array. It is currently more efficient and easier to use either the array get or array names, together with foreach, to iterate over all but very large arrays. See the examples below for how to do this.

array statistics arrayName
Returns statistics about the distribution of data within the hashtable that represents the array. This information includes the number of entries in the table, the number of buckets, and the utilization of the buckets.

array unset arrayName ?pattern?
Unsets all of the elements in the array that match pattern (using the matching rules of string match). If arrayName is not the name of an array variable or there are no matching elements in the array, no error will be raised. If pattern is omitted and arrayName is an array variable, then the command unsets the entire array. The command always returns an empty string.

**EXAMPLES**

```bash
array set colorcount {
    red 1
    green 5
    blue 4
    white 9
}

foreach {color count} [array get colorcount] {
    puts "Color: $color Count: $count"
}
```
foreach color [array names colorcount] {
  puts "Color: $color Count: $colorcount($color)"
}

array statistics colorcount

  → 4 entries in table, 4 buckets
  number of buckets with 0 entries: 1
  number of buckets with 1 entries: 2
  number of buckets with 2 entries: 1
  number of buckets with 3 entries: 0
  number of buckets with 4 entries: 0
  number of buckets with 5 entries: 0
  number of buckets with 6 entries: 0
  number of buckets with 7 entries: 0
  number of buckets with 8 entries: 0
  number of buckets with 9 entries: 0
  number of buckets with 10 or more entries: 0
  average search distance for entry: 1.2
SEE ALSO

list, string, variable, trace, foreach

KEYWORDS

array, element names, search

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

exit - End the application

SYNOPSIS

exit ?returnCode?

DESCRIPTION

Terminate the process, returning returnCode to the system as the exit status. If returnCode is not specified then it defaults to 0.

EXAMPLE

Since non-zero exit codes are usually interpreted as error cases by the calling process, the exit command is an important part of signaling that something fatal has gone wrong. This code fragment is useful in scripts to act as a general problem trap:

```tcl
proc main {} {
    # ... put the real main code in here ...
}

if {[catch {main} msg options]} {
    puts stderr "unexpected script error: $msg"
    if {[info exist env(DEBUG)]} {
        puts stderr "---- BEGIN TRACE ----"
        puts stderr [dict get $options -errorinfo]
        puts stderr "---- END TRACE ----"
    }
}
```
# Reserve code 1 for "expected" error exits...
exit 2

SEE ALSO

exec

KEYWORDS

exit, process

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

linsert - Insert elements into a list

SYNOPSIS

linsert list index element ?element element ...?

DESCRIPTION

This command produces a new list from list by inserting all of the element arguments just before the index' th element of list. Each element argument will become a separate element of the new list. If index is less than or equal to zero, then the new elements are inserted at the beginning of the list. The interpretation of the index value is the same as for the command string index, supporting simple index arithmetic and indices relative to the end of the list.

EXAMPLE

Putting some values into a list, first indexing from the start and then indexing from the end, and then chaining them together:

```
set oldList {the fox jumps over the dog}
set midList [linsert $oldList 1 quick]
set newList [linsert $midList end-1 lazy]
# The old lists still exist though...
set newerList [linsert [linsert $oldList end-1 quick]
```

SEE ALSO
list, lappend, lindex, llength, lsearch, lset, lsort, lrange, lreplace, string

KEYWORDS

element, insert, list

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 2001 Kevin B. Kenny <kennykb(at)acm.org>. All rights reserved.
NAME

puts - Write to a channel

SYNOPSIS

puts ?-nonewline? ?channelId? string

DESCRIPTION

Writes the characters given by string to the channel given by channelId.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdout or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension. The channel must have been opened for output.

If no channelId is specified then it defaults to stdout. Puts normally outputs a newline character after string, but this feature may be suppressed by specifying the -nonewline switch.

Newline characters in the output are translated by puts to platform-specific end-of-line sequences according to the current value of the -translation option for the channel (for example, on PCs newlines are normally replaced with carriage-return-linefeed sequences. See the fconfigure manual entry for a discussion on ways in which fconfigure will alter output.

Tcl buffers output internally, so characters written with puts may not appear immediately on the output file or device; Tcl will normally delay output until the buffer is full or the channel is closed. You can force
output to appear immediately with the \texttt{flush} command.

When the output buffer fills up, the \texttt{puts} command will normally block until all the buffered data has been accepted for output by the operating system. If \emph{channelId} is in nonblocking mode then the \texttt{puts} command will not block even if the operating system cannot accept the data. Instead, Tcl continues to buffer the data and writes it in the background as fast as the underlying file or device can accept it. The application must use the Tcl event loop for nonblocking output to work; otherwise Tcl never finds out that the file or device is ready for more output data. It is possible for an arbitrarily large amount of data to be buffered for a channel in nonblocking mode, which could consume a large amount of memory. To avoid wasting memory, nonblocking I/O should normally be used in an event-driven fashion with the \texttt{fileevent} command (do not invoke \texttt{puts} unless you have recently been notified via a file event that the channel is ready for more output data).

\subsection*{EXAMPLES}

Write a short message to the console (or wherever \texttt{stdout} is directed):

\begin{verbatim}
puts "Hello, World!"
\end{verbatim}

Print a message in several parts:

\begin{verbatim}
puts -nonewline "Hello, 
puts "World!"
\end{verbatim}

Print a message to the standard error channel:

\begin{verbatim}
puts stderr "Hello, World!"
\end{verbatim}

Append a log message to a file:
set chan [open my.log a]
set timestamp [clock format [clock seconds]]
puts $chan "$timestamp - Hello, World!"
close $chan

SEE ALSO

file, fileevent, Tcl_StandardChannels

KEYWORDS

channel, newline, output, write
NAME
expr - Evaluate an expression

SYNOPSIS

DESCRIPTION

OPERANDS

OPERATORS
- + ~ !
**
* / %
+-
<< >>
< > <= >=
== !=
eq ne
in ni
&
^|
&&
||
x?y:z

MATH FUNCTIONS

TYPES, OVERFLOW, AND PRECISION

STRING OPERATIONS

PERFORMANCE CONSIDERATIONS

EXAMPLES

SEE ALSO

KEYWORDS

COPYRIGHT
expr - Evaluate an expression

**SYNOPSIS**

```plaintext
expr arg ?arg arg ...?
```

**DESCRIPTION**

Concatenates `args` (adding separator spaces between them), evaluates the result as a Tcl expression, and returns the value. The operators permitted in Tcl expressions include a subset of the operators permitted in C expressions. For those operators common to both Tcl and C, Tcl applies the same meaning and precedence as the corresponding C operators. Expressions almost always yield numeric results (integer or floating-point values). For example, the expression

```plaintext
expr 8.2 + 6
```

evaluates to 14.2. Tcl expressions differ from C expressions in the way that operands are specified. Also, Tcl expressions support non-numeric operands and string comparisons, as well as some additional operators not found in C.

**OPERANDS**

A Tcl expression consists of a combination of operands, operators, and parentheses. White space may be used between the operands and operators and parentheses; it is ignored by the expression's instructions. Where possible, operands are interpreted as integer values. Integer values may be specified in decimal (the normal case), in binary (if the first two characters of the operand are `0b`), in octal (if the first two characters of the operand are `0o`), or in hexadecimal (if the first two characters of the operand are `0x`). For compatibility with older Tcl releases, an octal integer value is also indicated simply when the first character of the operand is `0`, whether or not the second character is also `o`. If an operand does not have one of the integer formats given...
above, then it is treated as a floating-point number if that is possible. Floating-point numbers may be specified in any of several common formats making use of the decimal digits, the decimal point ., the characters e or E indicating scientific notation, and the sign characters + or -. For example, all of the following are valid floating-point numbers: 2.1, 3., 6e4, 7.91e+16. Also recognized as floating point values are the strings Inf and NaN making use of any case for each character. If no numeric interpretation is possible (note that all literal operands that are not numeric or boolean must be quoted with either braces or with double quotes), then an operand is left as a string (and only a limited set of operators may be applied to it).

Operands may be specified in any of the following ways:

[1] As a numeric value, either integer or floating-point.

[2] As a boolean value, using any form understood by string is boolean.

[3] As a Tcl variable, using standard $ notation. The variable's value will be used as the operand.

[4] As a string enclosed in double-quotes. The expression parser will perform backslash, variable, and command substitutions on the information between the quotes, and use the resulting value as the operand.

[5] As a string enclosed in braces. The characters between the open brace and matching close brace will be used as the operand without any substitutions.

[6] As a Tcl command enclosed in brackets. The command will be
executed and its result will be used as the operand.

[7]  
As a mathematical function whose arguments have any of the above forms for operands, such as \( \sin(x) \). See MATH FUNCTIONS below for a discussion of how mathematical functions are handled.

Where the above substitutions occur (e.g. inside quoted strings), they are performed by the expression's instructions. However, the command parser may already have performed one round of substitution before the expression processor was called. As discussed below, it is usually best to enclose expressions in braces to prevent the command parser from performing substitutions on the contents.

For some examples of simple expressions, suppose the variable \( a \) has the value 3 and the variable \( b \) has the value 6. Then the command on the left side of each of the lines below will produce the value on the right side of the line:

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr 3.1 + $a 6.1</td>
<td>9.1</td>
</tr>
<tr>
<td>expr 2 + &quot;$a.$b&quot; 5.6</td>
<td>7.6</td>
</tr>
<tr>
<td>expr 4*[llength &quot;6 2&quot;] 8</td>
<td>8</td>
</tr>
<tr>
<td>expr {word one} &lt; &quot;word $a&quot;} 0</td>
<td>0</td>
</tr>
</tbody>
</table>

OPERATORS

The valid operators (most of which are also available as commands in the tcl::mathop namespace; see the mathop(n) manual page for details) are listed below, grouped in decreasing order of precedence:

- + ~ !

Unary minus, unary plus, bit-wise NOT, logical NOT. None of these operators may be applied to string operands, and bit-wise NOT may be applied only to integers.
** Exponentiation. Valid for any numeric operands.

* / %
Multiply, divide, remainder. None of these operators may be applied to string operands, and remainder may be applied only to integers. The remainder will always have the same sign as the divisor and an absolute value smaller than the divisor.

+ -
Add and subtract. Valid for any numeric operands.

<< >>
Left and right shift. Valid for integer operands only. A right shift always propagates the sign bit.

< > <= >=
Boolean less, greater, less than or equal, and greater than or equal. Each operator produces 1 if the condition is true, 0 otherwise. These operators may be applied to strings as well as numeric operands, in which case string comparison is used.

== !=
Boolean equal and not equal. Each operator produces a zero/one result. Valid for all operand types.

eq ne
Boolean string equal and string not equal. Each operator produces a zero/one result. The operand types are interpreted only as strings.

in ni
List containment and negated list containment. Each operator produces a zero/one result and treats its first argument as a string and its second argument as a Tcl list. The in operator indicates whether the first argument is a member of the second argument list; the ni operator inverts the sense of the result.
Bit-wise AND. Valid for integer operands only.

Bit-wise exclusive OR. Valid for integer operands only.

Bit-wise OR. Valid for integer operands only.

Logical AND. Produces a 1 result if both operands are non-zero, 0 otherwise. Valid for boolean and numeric (integers or floating-point) operands only.

Logical OR. Produces a 0 result if both operands are zero, 1 otherwise. Valid for boolean and numeric (integers or floating-point) operands only.

If-then-else, as in C. If \( x \) evaluates to non-zero, then the result is the value of \( y \). Otherwise the result is the value of \( z \). The \( x \) operand must have a boolean or numeric value.

See the C manual for more details on the results produced by each operator. The exponentiation operator promotes types like the multiply and divide operators, and produces a result that is the same as the output of the \texttt{pow} function (after any type conversions.) All of the binary operators group left-to-right within the same precedence level. For example, the command

```
expr {4*2 < 7}
```

returns 0.

The &&, ||, and ?: operators have “lazy evaluation”, just as in C, which means that operands are not evaluated if they are not needed to
determine the outcome. For example, in the command

```
expr \{\$v \? [a] : [b]\}
```

only one of “[a]” or “[b]” will actually be evaluated, depending on the value of \$v. Note, however, that this is only true if the entire expression is enclosed in braces; otherwise the Tcl parser will evaluate both “[a]” and “[b]” before invoking the expr command.

**MATH FUNCTIONS**

When the expression parser encounters a mathematical function such as \texttt{sin($x)}}, it replaces it with a call to an ordinary Tcl function in the \texttt{tcl::mathfunc} namespace. The processing of an expression such as:

```
expr \{sin($x+$y)\}
```

is the same in every way as the processing of:

```
expr \{[tcl::mathfunc::sin [expr \{$x+$y\}]]\}
```

which in turn is the same as the processing of:

```
tcl::mathfunc::sin [expr \{$x+$y\}]
```

The executor will search for \texttt{tcl::mathfunc::sin} using the usual rules for resolving functions in namespaces. Either \texttt{::tcl::mathfunc::sin} or \texttt{[namespace current]::tcl::mathfunc::sin} will satisfy the request, and others may as well (depending on the current \texttt{namespace path} setting).
See the `mathfunc(n)` manual page for the math functions that are available by default.

**TYPES, OVERFLOW, AND PRECISION**

All internal computations involving integers are done calling on the LibTomMath multiple precision integer library as required so that all integer calculations are performed exactly. Note that in Tcl releases prior to 8.5, integer calculations were performed with one of the C types `long int` or `Tcl_WideInt`, causing implicit range truncation in those calculations where values overflowed the range of those types. Any code that relied on these implicit truncations will need to explicitly add `int()` or `wide()` function calls to expressions at the points where such truncation is required to take place.

All internal computations involving floating-point are done with the C type `double`. When converting a string to floating-point, exponent overflow is detected and results in the `double` value of `Inf` or `-Inf` as appropriate. Floating-point overflow and underflow are detected to the degree supported by the hardware, which is generally pretty reliable.

Conversion among internal representations for integer, floating-point, and string operands is done automatically as needed. For arithmetic computations, integers are used until some floating-point number is introduced, after which floating-point is used. For example,

```tcl
expr {5 / 4}
```

returns 1, while

```tcl
expr {5 / 4.0}
expr {5 / ( [string length "abcd"] + 0.0 )}
```

both return 1.25. Floating-point values are always returned with a “.” or
an “e” so that they will not look like integer values. For example,

```
expr {20.0/5.0}
```

returns 4.0, not 4.

### STRING OPERATIONS

String values may be used as operands of the comparison operators, although the expression evaluator tries to do comparisons as integer or floating-point when it can, except in the case of the `eq` and `ne` operators. If one of the operands of a comparison is a string and the other has a numeric value, a canonical string representation of the numeric operand value is generated to compare with the string operand. Canonical string representation for integer values is a decimal string format. Canonical string representation for floating-point values is that produced by the `%g` format specifier of Tcl's `format` command. For example, the commands

```
expr {"0x03" > "2"}
expr {"0y" < "0x12"}
```

both return 1. The first comparison is done using integer comparison, and the second is done using string comparison after the second operand is converted to the string 18. Because of Tcl's tendency to treat values as numbers whenever possible, it is not generally a good idea to use operators like `==` when you really want string comparison and the values of the operands could be arbitrary; it is better in these cases to use the `eq` or `ne` operators, or the `string` command instead.

### PERFORMANCE CONSIDERATIONS

Enclose expressions in braces for the best speed and the smallest storage requirements. This allows the Tcl bytecode compiler to generate
the best code.

As mentioned above, expressions are substituted twice: once by the Tcl parser and once by the `expr` command. For example, the commands

```
set a 3
set b {a + 2}
expr $b*4
```

return 11, not a multiple of 4. This is because the Tcl parser will first substitute `$a + 2` for the variable `b`, then the `expr` command will evaluate the expression `$a + 2*4`.

Most expressions do not require a second round of substitutions. Either they are enclosed in braces or, if not, their variable and command substitutions yield numbers or strings that do not themselves require substitutions. However, because a few unbraced expressions need two rounds of substitutions, the bytecode compiler must emit additional instructions to handle this situation. The most expensive code is required for unbraced expressions that contain command substitutions. These expressions must be implemented by generating new code each time the expression is executed. When the expression is unbraced to allow the substitution of a function or operator, consider using the commands documented in the `mathfunc(n)` or `mathop(n)` manual pages directly instead.

**EXAMPLES**

Define a procedure that computes an “interesting” mathematical function:

```
proc tcl::mathfunc::calc {x y} {
    expr { (x**2 - y**2) / exp(x**2 + y**2) }
}
```
Convert polar coordinates into cartesian coordinates:

```bash
# convert from ($radius,$angle)
set x [expr { $radius * cos($angle) }]
set y [expr { $radius * sin($angle) }]
```

Convert cartesian coordinates into polar coordinates:

```bash
# convert from ($x,$y)
set radius [expr { hypot($y, $x) }]
set angle [expr { atan2($y, $x) }]
```

Print a message describing the relationship of two string values to each other:

```bash
puts "a and b are [expr {$a eq $b ? {equal} : {different}}]
```

Set a variable to whether an environment variable is both defined at all and also set to a true boolean value:

```bash
set isTrue [expr {
    [info exists ::env(SOME_ENV_VAR)] &&
    [string is true -strict ::env(SOME_ENV_VAR)]
}]
```

Generate a random integer in the range 0..99 inclusive:

```bash
set randNum [expr { int(100 * rand()) }]
```
SEE ALSO

array, for, if, mathfunc, mathop, namespace, proc, string, Tcl, while

KEYWORDS

arithmetic, boolean, compare, expression, fuzzy comparison

COPYRIGHT

Copyright (c) 1993 The Regents of the University of California.  
Copyright (c) 1994-2000 Sun Microsystems Incorporated.  
Copyright (c) 2005 by Kevin B. Kenny <kennykb@acm.org>. All rights reserved.

Copyright © 1995-1997 Roger E. Critchlow Jr.  
Copyright © 1993 The Regents of the University of California.  
Copyright © 1994-2000 Sun Microsystems, Inc.  
Copyright © 2005 by Kevin B. Kenny <kennykb(at)acm.org>. All rights reserved
NAME

list - Create a list

SYNOPSIS

list ?arg arg ...?

DESCRIPTION

This command returns a list comprised of all the args, or an empty string if no args are specified. Braces and backslashes get added as necessary, so that the lindex command may be used on the result to re-extract the original arguments, and also so that eval may be used to execute the resulting list, with arg1 comprising the command's name and the other args comprising its arguments. List produces slightly different results than concat: concat removes one level of grouping before forming the list, while list works directly from the original arguments.

EXAMPLE

The command

```
list a b "c d e " " f {g h}"
```

will return

```
a b {c d e } { f {g h}}
```
while `concat` with the same arguments will return

```
a b c d e f {g h}
```

**SEE ALSO**

`lappend`, `lindex`, `linsert`, `llength`, `lrange`, `lrepeat`, `lreplace`, `lsearch`, `lset`, `lsort`

**KEYWORDS**

`element`, `list`
NAME

pwd - Return the absolute path of the current working directory

SYNOPSIS

pwd

DESCRIPTION

Returns the absolute path name of the current working directory.

EXAMPLE

Sometimes it is useful to change to a known directory when running some external command using exec, but it is important to keep the application usually running in the directory that it was started in (unless the user specifies otherwise) since that minimizes user confusion. The way to do this is to save the current directory while the external command is being run:

```
set tarFile [file normalize somefile.tar]
set savedDir [pwd]
cd /tmp
exec tar -xf $tarFile
cd $savedDir
```

SEE ALSO

file, cd, glob, filename
<table>
<thead>
<tr>
<th>KEYWORDS</th>
</tr>
</thead>
</table>

**working directory**
NAME

fblocked - Test whether the last input operation exhausted all available input

SYNOPSIS

fblocked channelId

DESCRIPTION

The fblocked command returns 1 if the most recent input operation on channelId returned less information than requested because all available input was exhausted. For example, if gets is invoked when there are only three characters available for input and no end-of-line sequence, gets returns an empty string and a subsequent call to fblocked will return 1.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

EXAMPLE

The fblocked command is particularly useful when writing network servers, as it allows you to write your code in a line-by-line style without preventing the servicing of other connections. This can be seen in this simple echo-service:

```
# This is called whenever a new client connects to t
```
proc connect {chan host port} {
    set clientName [format <%s:%d> $host $port]
    puts "connection from $clientName"
    fconfigure $chan -blocking 0 -buffering line
    fileevent $chan readable [list echoLine $chan $clientName]
}

# This is called whenever either at least one byte of data is available, or
# the channel was closed by the client.
proc echoLine {chan clientName} {
    gets $chan line
    if {[eof $chan]} {
        puts "finishing connection from $clientName"
        close $chan
    } elseif {![fblocked $chan]} {
        # Didn't block waiting for end-of-line
        puts "$clientName - $line"
        puts $chan $line
    }
}

# Create the server socket and enter the event-loop
# for incoming connections...
socket -server connect 12345
vwait forever
NAME

llength - Count the number of elements in a list

SYNOPSIS

llength list

DESCRIPTION

Treats list as a list and returns a decimal string giving the number of elements in it.

EXAMPLES

The result is the number of elements:

```
% llength {a b c d e}  
5
% llength {a b c}     
3
% llength {}          
0
```

Elements are not guaranteed to be exactly words in a dictionary sense of course, especially when quoting is used:

```
% llength {a b {c d} e} 
4
% llength {a b { } c d e}
```
An empty list is not necessarily an empty string:

% set var { }; puts "[string length $var],[llength $var]
1,0

SEE ALSO

list, lappend, lindex, linsert, lsearch, lset, lsort, lrange, lreplace

KEYWORDS

element, list, length
NAME
  re_syntax - Syntax of Tcl regular expressions

DESCRIPTION

DIFFERENT FLAVORS OF REs

REGULAR EXPRESSION SYNTAX

QUANTIFIERS

*    
+    
?    
{m}  
{m,} 
{m,n} 
*?  +?  ??  {m}?  {m,}?  {m,n}? 

ATOMS

(re) 
(?:re) 
() 
(?:) 
[chars] 
, 
\k 
\c 
{ 
} 

CONSTRAINTS

^    
$    
(?=re) 
(?!re) 

BRACKET EXPRESSIONS

CHARACTER CLASSES
alpha
upper
lower
digit
xdigit
alnum
print
blank
space
punct
graph
cntrl

BRACKETED CONSTRAINTS

COLLATING ELEMENTS

EQUIVALENCE CLASSES

ESCAPES

CHARACTER-ENTRY ESCAPES
\a
\b
\B
\cX
\e
\f
\n
\r
\t
\u
\wxyz
\U
\us
\u
\x
\hhh
\0
\xy
\xyz

CLASS-SHORTHAND ESCAPES
\d
\s
\w
<table>
<thead>
<tr>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>re_syntax - Syntax of Tcl regular expressions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>constraint escapes</td>
</tr>
<tr>
<td>back references</td>
</tr>
<tr>
<td>metasyntax</td>
</tr>
<tr>
<td>matching</td>
</tr>
<tr>
<td>limits and compatibility</td>
</tr>
<tr>
<td>basic regular expressions</td>
</tr>
<tr>
<td>see also</td>
</tr>
<tr>
<td>keywords</td>
</tr>
</tbody>
</table>
A regular expression describes strings of characters. It's a pattern that matches certain strings and does not match others.

### DIFFERENT FLAVORS OF REs

Regular expressions ("RE"s), as defined by POSIX, come in two flavors: extended REs ("ERE"s) and basic REs ("BRE"s). EREs are roughly those of the traditional egrep, while BREs are roughly those of the traditional ed. This implementation adds a third flavor, advanced REs ("ARE"s), basically EREs with some significant extensions.

This manual page primarily describes AREs. BREs mostly exist for backward compatibility in some old programs; they will be discussed at the end. POSIX EREs are almost an exact subset of AREs. Features of AREs that are not present in EREs will be indicated.

### REGULAR EXPRESSION SYNTAX

Tcl regular expressions are implemented using the package written by Henry Spencer, based on the 1003.2 spec and some (not quite all) of the Perl5 extensions (thanks, Henry!). Much of the description of regular expressions below is copied verbatim from his manual entry.

An ARE is one or more branches, separated by "|", matching anything that matches any of the branches.

A branch is zero or more constraints or quantified atoms, concatenated. It matches a match for the first, followed by a match for the second, etc; an empty branch matches the empty string.

### QUANTIFIERS

A quantified atom is an atom possibly followed by a single quantifier. Without a quantifier, it matches a single match for the atom. The quantifiers, and what a so-quantified atom matches, are:

* a sequence of 0 or more matches of the atom
+ 
  a sequence of 1 or more matches of the atom

? 
  a sequence of 0 or 1 matches of the atom

{m} 
  a sequence of exactly m matches of the atom

{m,} 
  a sequence of m or more matches of the atom

{m,n} 
  a sequence of m through n (inclusive) matches of the atom; m may not exceed n

*? +? ?? {m}? {m,}? {m,n}? 
  non-greedy quantifiers, which match the same possibilities, but prefer the smallest number rather than the largest number of matches (see MATCHING)

The forms using { and } are known as bounds. The numbers m and n are unsigned decimal integers with permissible values from 0 to 255 inclusive.

**ATOMS**

An atom is one of:

(re) 
  matches a match for re (re is any regular expression) with the match noted for possible reporting

(?:re) 
  as previous, but does no reporting (a “non-capturing” set of parentheses)

()
matches an empty string, noted for possible reporting

(?): matches an empty string, without reporting

[chars] a *bracket expression*, matching any one of the *chars* (see *BRACKET EXPRESSIONS* for more detail)

. matches any single character

\k matches the non-alphanumeric character *k* taken as an ordinary character, e.g. \\ matches a backslash character

\c where *c* is alphanumeric (possibly followed by other characters), an escape (AREs only), see *ESCAPES* below

{ when followed by a character other than a digit, matches the left-brace character "{"; when followed by a digit, it is the beginning of a bound (see above)

\x where *x* is a single character with no other significance, matches that character.

---

**CONSTRAINTS**

A *constraint* matches an empty string when specific conditions are met. A constraint may not be followed by a quantifier. The simple constraints are as follows; some more constraints are described later, under *ESCAPES*.

^ matches at the beginning of a line
matches at the end of a line

(?=re)
positive lookahead (AREs only), matches at any point where a substring matching re begins

(?!re)
negative lookahead (AREs only), matches at any point where no substring matching re begins

The lookahead constraints may not contain back references (see later), and all parentheses within them are considered non-capturing.

An RE may not end with “\".

### BRACKET EXPRESSIONS

A bracket expression is a list of characters enclosed in “[ ]”. It normally matches any single character from the list (but see below). If the list begins with “^”, it matches any single character (but see below) not from the rest of the list.

If two characters in the list are separated by “-“, this is shorthand for the full range of characters between those two (inclusive) in the collating sequence, e.g. “[0-9]” in Unicode matches any conventional decimal digit. Two ranges may not share an endpoint, so e.g. “a-c-e” is illegal. Ranges in Tcl always use the Unicode collating sequence, but other programs may use other collating sequences and this can be a source of incompatibility between programs.

To include a literal ] or - in the list, the simplest method is to enclose it in [. and .] to make it a collating element (see below). Alternatively, make it the first character (following a possible “^”), or (AREs only) precede it with “\". Alternatively, for “-”, make it the last character, or the second endpoint of a range. To use a literal - as the first endpoint of a range, make it a collating element or (AREs only) precede it with “\". With the exception of these, some combinations using [ (see next paragraphs),
and escapes, all other special characters lose their special significance within a bracket expression.

**CHARACTER CLASSES**

Within a bracket expression, the name of a *character class* enclosed in [: and :] stands for the list of all characters (not all collating elements!) belonging to that class. Standard character classes are:

- **alpha**
  A letter.

- **upper**
  An upper-case letter.

- **lower**
  A lower-case letter.

- **digit**
  A decimal digit.

- **xdigit**
  A hexadecimal digit.

- **alnum**
  An alphanumerical (letter or digit).

- **print**
  A "printable" (same as graph, except also including space).

- **blank**
  A space or tab character.

- **space**
  A character producing white space in displayed text.

- **punct**
  A punctuation character.
A character with a visible representation (includes both alnum and punct).

A control character.

A locale may provide others. A character class may not be used as an endpoint of a range.

(Note: the current Tcl implementation has only one locale, the Unicode locale, which supports exactly the above classes.)

**BRACKETED CONSTRAINTS**

There are two special cases of bracket expressions: the bracket expressions “[[[:<:]]]” and “[[[:>:]]]” are constraints, matching empty strings at the beginning and end of a word respectively. A word is defined as a sequence of word characters that is neither preceded nor followed by word characters. A word character is an alnum character or an underscore (“_”). These special bracket expressions are deprecated; users of AREs should use constraint escapes instead (see below).

**COLLATING ELEMENTS**

Within a bracket expression, a collating element (a character, a multi-character sequence that collates as if it were a single character, or a collating-sequence name for either) enclosed in [ and ] stands for the sequence of characters of that collating element. The sequence is a single element of the bracket expression's list. A bracket expression in a locale that has multi-character collating elements can thus match more than one character. So (insidiously), a bracket expression that starts with ^ can match multi-character collating elements even if none of them appear in the bracket expression!

(Note: Tcl has no multi-character collating elements. This information is only for illustration.)
For example, assume the collating sequence includes a ch multi-character collating element. Then the RE “\[.ch.]\*c” (zero or more “chs” followed by “c”) matches the first five characters of “chchccc”. Also, the RE “[^c]b” matches all of “chb” (because “[^c]” matches the multi-character “ch”).

### EQUIVALENCE CLASSES

Within a bracket expression, a collating element enclosed in [= and =] is an equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself. (If there are no other equivalent collating elements, the treatment is as if the enclosing delimiters were “[.” and “.].”) For example, if o and ô are the members of an equivalence class, then “[=o=]”, “[=ô=]”, and “[ôô]” are all synonymous. An equivalence class may not be an endpoint of a range.

(Note: Tcl implements only the Unicode locale. It does not define any equivalence classes. The examples above are just illustrations.)

### ESCAPES

Escapes (AREs only), which begin with a \ followed by an alphanumeric character, come in several varieties: character entry, class shorthands, constraint escapes, and back references. A \ followed by an alphanumeric character but not constituting a valid escape is illegal in AREs. In EREs, there are no escapes: outside a bracket expression, a \ followed by an alphanumeric character merely stands for that character as an ordinary character, and inside a bracket expression, \ is an ordinary character. (The latter is the one actual incompatibility between EREs and AREs.)

### CHARACTER-ENTRY ESCAPES

Character-entry escapes (AREs only) exist to make it easier to specify non-printing and otherwise inconvenient characters in REs:

\a
alert (bell) character, as in C

\b
backspace, as in C

\B
synonym for \ to help reduce backslash doubling in some applications where there are multiple levels of backslash processing

\cX
(where X is any character) the character whose low-order 5 bits are the same as those of X, and whose other bits are all zero

\e
the character whose collating-sequence name is “ESC”, or failing that, the character with octal value 033

\f
formfeed, as in C

\n
newline, as in C

\r
carriage return, as in C

\t
horizontal tab, as in C

\uwxyz
(where wxyz is exactly four hexadecimal digits) the Unicode character U+wxyz in the local byte ordering

\Ustuvwxyz
(where stuvwxyz is exactly eight hexadecimal digits) reserved for a somewhat-hypothetical Unicode extension to 32 bits

\v
vertical tab, as in C are all available.

\xhhhh
(where \texttt{hhh} is any sequence of hexadecimal digits) the character whose hexadecimal value is \texttt{0xhh\texttt{h}} (a single character no matter how many hexadecimal digits are used).

\0
the character whose value is \texttt{0}

\xy
(\texttt{xy} is exactly two octal digits, and is not a back reference (see below)) the character whose octal value is \texttt{0\texttt{xy}}

\xyz
(\texttt{xyz} is exactly three octal digits, and is not a back reference (see below)) the character whose octal value is \texttt{0\texttt{xyz}}

Hexadecimal digits are "\texttt{0-9}, \texttt{a-f}" , and "\texttt{A-F}". Octal digits are "\texttt{0-7}".

The character-entry escapes are always taken as ordinary characters. For example, \texttt{\135} is ] in Unicode, but \texttt{\135} does not terminate a bracket expression. Beware, however, that some applications (e.g., C compilers and the Tcl interpreter if the regular expression is not quoted with braces) interpret such sequences themselves before the regular-expression package gets to see them, which may require doubling (quadrupling, etc.) the "\texttt{\textbackslash}".

**CLASS-SHORTHAND ESCAPES**

Class-shorthand escapes (AREs only) provide shorthands for certain commonly-used character classes:

\d
\[[:digit:]\]

\s
\[[:space:]\]
Within bracket expressions, “\d”, “\s”, and “\w” lose their outer brackets, and “\D”, “\S”, and “\W” are illegal. (So, for example, “[a-c\d]” is equivalent to “[a-c[:digit:]]”. Also, “[a-c\D]”, which is equivalent to “[a-c^[:digit:]]”, is illegal.)

**CONSTRAINT ESCAPES**

A constraint escape (AREs only) is a constraint, matching the empty string if specific conditions are met, written as an escape:

\A
matches only at the beginning of the string (see MATCHING, below, for how this differs from “^”)

\m
matches only at the beginning of a word

\M
matches only at the end of a word

\y
matches only at the beginning or end of a word

\Y
matches only at a point that is not the beginning or end of a word

\Z
matches only at the end of the string (see MATCHING, below, for how this differs from “$”)

\m
(\text{where } m \text{ is a nonzero digit}) \text{ a } \textit{back reference}, \text{ see below}

\mnn
(\text{where } m \text{ is a nonzero digit, and } nn \text{ is some more digits, and the decimal value } mnn \text{ is not greater than the number of closing capturing parentheses seen so far}) \text{ a } \textit{back reference}, \text{ see below}

A word is defined as in the specification of “\([[:<:]]\)” and “\([[:>:]]\)” above. Constraint escapes are illegal within bracket expressions.

**BACK REFERENCES**

A back reference (AREs only) matches the same string matched by the parenthesized subexpression specified by the number, so that (e.g.) “\((\texttt{bc})\m1\)” matches “bb” or “cc” but not “bc”. The subexpression must entirely precede the back reference in the RE. Subexpressions are numbered in the order of their leading parentheses. Non-capturing parentheses do not define subexpressions.

There is an inherent historical ambiguity between octal character-entry escapes and back references, which is resolved by heuristics, as hinted at above. A leading zero always indicates an octal escape. A single non-zero digit, not followed by another digit, is always taken as a back reference. A multi-digit sequence not starting with a zero is taken as a back reference if it comes after a suitable subexpression (i.e. the number is in the legal range for a back reference), and otherwise is taken as octal.

**METASYNTAX**

In addition to the main syntax described above, there are some special forms and miscellaneous syntactic facilities available.

Normally the flavor of RE being used is specified by application-
dependent means. However, this can be overridden by a director. If an RE of any flavor begins with “***:”, the rest of the RE is an ARE. If an RE of any flavor begins with “***=”, the rest of the RE is taken to be a literal string, with all characters considered ordinary characters.

An ARE may begin with embedded options: a sequence (?xyz) (where xyz is one or more alphabetic characters) specifies options affecting the rest of the RE. These supplement, and can override, any options specified by the application. The available option letters are:

- **b**: rest of RE is a BRE
- **c**: case-sensitive matching (usual default)
- **e**: rest of RE is an ERE
- **i**: case-insensitive matching (see MATCHING, below)
- **m**: historical synonym for **n**
- **n**: newline-sensitive matching (see MATCHING, below)
- **p**: partial newline-sensitive matching (see MATCHING, below)
- **q**: rest of RE is a literal (“quoted”) string, all ordinary characters
- **s**: non-newline-sensitive matching (usual default)
- **t**: tight syntax (usual default; see below)
**w**
inverse partial newline-sensitive ("weird") matching (see MATCHING, below)

**x**
expanded syntax (see below)

Embedded options take effect at the ) terminating the sequence. They are available only at the start of an ARE, and may not be used later within it.

In addition to the usual (tight) RE syntax, in which all characters are significant, there is an expanded syntax, available in all flavors of RE with the -expanded switch, or in AREs with the embedded x option. In the expanded syntax, white-space characters are ignored and all characters between a # and the following newline (or the end of the RE) are ignored, permitting paragraphing and commenting a complex RE. There are three exceptions to that basic rule:

- a white-space character or "#" preceded by "\" is retained
- white space or "#" within a bracket expression is retained
- white space and comments are illegal within multi-character symbols like the ARE "(?::" or the BRE "\"

Expanded-syntax white-space characters are blank, tab, newline, and any character that belongs to the space character class.

Finally, in an ARE, outside bracket expressions, the sequence "(?#tti)" (where tti is any text not containing a ") is a comment, completely ignored. Again, this is not allowed between the characters of multi-character symbols like "(?::. Such comments are more a historical artifact than a useful facility, and their use is deprecated; use the expanded syntax instead.

None of these metasyntax extensions is available if the application (or an initial "***=" director) has specified that the user's input be treated as a literal string rather than as an RE.
In the event that an RE could match more than one substring of a given string, the RE matches the one starting earliest in the string. If the RE could match more than one substring starting at that point, its choice is determined by its preference: either the longest substring, or the shortest.

Most atoms, and all constraints, have no preference. A parenthesized RE has the same preference (possibly none) as the RE. A quantified atom with quantifier \{m\} or \{m\}? has the same preference (possibly none) as the atom itself. A quantified atom with other normal quantifiers (including \{m,n\} with \(m\) equal to \(n\)) prefers longest match. A quantified atom with other non-greedy quantifiers (including \{m,n\}? with \(m\) equal to \(n\)) prefers shortest match. A branch has the same preference as the first quantified atom in it which has a preference. An RE consisting of two or more branches connected by the | operator prefers longest match.

Subject to the constraints imposed by the rules for matching the whole RE, subexpressions also match the longest or shortest possible substrings, based on their preferences, with subexpressions starting earlier in the RE taking priority over ones starting later. Note that outer subexpressions thus take priority over their component subexpressions.

Note that the quantifiers \{1,1\} and \{1,1\}? can be used to force longest and shortest preference, respectively, on a subexpression or a whole RE.

Match lengths are measured in characters, not collating elements. An empty string is considered longer than no match at all. For example, “bb*” matches the three middle characters of “abbbc”, “(week|wee) (night|knights)” matches all ten characters of “weeknights”, when “(.*)*” is matched against “abc” the parenthesized subexpression matches all three characters, and when “(a*)*” is matched against “bc” both the whole RE and the parenthesized subexpression match an empty string.
If case-independent matching is specified, the effect is much as if all case distinctions had vanished from the alphabet. When an alphabetic that exists in multiple cases appears as an ordinary character outside a bracket expression, it is effectively transformed into a bracket expression containing both cases, so that \texttt{x} becomes \texttt{[xX]}. When it appears inside a bracket expression, all case counterparts of it are added to the bracket expression, so that \texttt{[x]} becomes \texttt{[xX]} and \texttt{[^x]} becomes \texttt{[^xX]}.

If newline-sensitive matching is specified, . and bracket expressions using ^ will never match the newline character (so that matches will never cross newlines unless the RE explicitly arranges it) and ^ and $ will match the empty string after and before a newline respectively, in addition to matching at beginning and end of string respectively. ARE \texttt{\A} and \texttt{\Z} continue to match beginning or end of string only.

If partial newline-sensitive matching is specified, this affects . and bracket expressions as with newline-sensitive matching, but not ^ and $.

If inverse partial newline-sensitive matching is specified, this affects ^ and $ as with newline-sensitive matching, but not . and bracket expressions. This is not very useful but is provided for symmetry.

**LIMITS AND COMPATIBILITY**

No particular limit is imposed on the length of REs. Programs intended to be highly portable should not employ REs longer than 256 bytes, as a POSIX-compliant implementation can refuse to accept such REs.

The only feature of AREs that is actually incompatible with POSIX EREs is that \texttt{\} does not lose its special significance inside bracket expressions. All other ARE features use syntax which is illegal or has undefined or unspecified effects in POSIX EREs; the *** syntax of directors likewise is outside the POSIX syntax for both BREs and EREs.

Many of the ARE extensions are borrowed from Perl, but some have been changed to clean them up, and a few Perl extensions are not
present. Incompatibilities of note include “\b”, “\B”, the lack of special
treatment for a trailing newline, the addition of complemented bracket
expressions to the things affected by newline-sensitive matching, the
restrictions on parentheses and back references in lookahead
constraints, and the longest/shortest-match (rather than first-match)
matching semantics.

The matching rules for REs containing both normal and non-greedy
quantifiers have changed since early beta-test versions of this package.
(The new rules are much simpler and cleaner, but do not work as hard
at guessing the user's real intentions.)

Henry Spencer's original 1986 regexp package, still in widespread use
(e.g., in pre-8.1 releases of Tcl), implemented an early version of
today's EREs. There are four incompatibilities between regexp's near-
EREs ("RREs" for short) and AREs. In roughly increasing order of
significance:

• In AREs, \ followed by an alphanumeric character is either an
  escape or an error, while in RREs, it was just another way of writing
  the alphanumeric. This should not be a problem because there was
  no reason to write such a sequence in RREs.

• { followed by a digit in an ARE is the beginning of a bound, while
  in RREs, { was always an ordinary character. Such sequences
  should be rare, and will often result in an error because following
  characters will not look like a valid bound.

• In AREs, \ remains a special character within "[ ]", so a literal \n  within [ ] must be written "\\". \ also gives a literal \ within [ ] in
  RREs, but only truly paranoid programmers routinely doubled the
  backslash.

• AREs report the longest/shortest match for the RE, rather than
  the first found in a specified search order. This may affect some
  RREs which were written in the expectation that the first match
  would be reported. (The careful crafting of RREs to optimize the
  search order for fast matching is obsolete (AREs examine all
possible matches in parallel, and their performance is largely insensitive to their complexity) but cases where the search order was exploited to deliberately find a match which was not the longest/shortest will need rewriting.

**BASIC REGULAR EXPRESSIONS**

BREs differ from EREs in several respects. “|”, “+”, and ? are ordinary characters and there is no equivalent for their functionality. The delimiters for bounds are \{ and “\}”, with { and } by themselves ordinary characters. The parentheses for nested subexpressions are \( and “\)”, with ( and ) by themselves ordinary characters. ^ is an ordinary character except at the beginning of the RE or the beginning of a parenthesized subexpression, $ is an ordinary character except at the end of the RE or the end of a parenthesized subexpression, and * is an ordinary character if it appears at the beginning of the RE or the beginning of a parenthesized subexpression (after a possible leading “^”). Finally, single-digit back references are available, and \< and \> are synonyms for “[[<:]]” and “[[>:]]” respectively; no other escapes are available.

**SEE ALSO**

RegExp, regexp, regsub, lsearch, switch, text

**KEYWORDS**

match, regular expression, string

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1998 Sun Microsystems, Inc.
Copyright © 1999 Scripatics Corporation
NAME
tcltest - Test harness support code and utilities

SYNOPSIS

DESCRIPTION

COMMANDS

  test name description ?option value ...?
  test name description ?constraints? body result
  loadTestedCommands
  makeFile contents name ?directory?
  removeFile name ?directory?
  makeDirectory name ?directory?
  removeDirectory name ?directory?
  viewFile file ?directory?
  cleanupTests
  runAllTests

CONFIGURATION COMMANDS

  configure
  configure option
  configure option value ?option value ...?
  customMatch mode script
  testConstraint constraint ?boolean?
  interpreter ?executableName?
  outputChannel ?channelID?
  errorChannel ?channelID?

SHORTCUT COMMANDS

  debug ?level?
  errorFile ?filename?
  limitConstraints ?boolean?
  loadFile ?filename?
  loadScript ?script?
  match ?patternList?
matchDirectories ?patternList?
matchFiles ?patternList?
outputFile ?filename?
preserveCore ?level?
singleProcess ?boolean?
skip ?patternList?
skipDirectories ?patternList?
skipFiles ?patternList?
temporaryDirectory ?directory?
testsDirectory ?directory?
verbose ?level?

OTHER COMMANDS

test name description optionList
workingDirectory ?directoryName?
normalizeMsg msg
normalizePath pathVar
bytestring string

TESTS

-constraints keywordList|expression
-setup script
-body script
-cleanup script
-match mode
-result expectedValue
-output expectedValue
-errorOutput expectedValue
-returnCodes expectedCodeList

TEST CONSTRANTS

singleTestInterp
unix
win
nt
95
98
mac
unixOrWin
macOrWin
macOrUnix
tempNotWin
tempNotMac
unixCrash
winCrash
macCrash
emptyTest
knownBug
nonPortable
userInteraction
interactive
nonBlockFiles
asyncPipeClose
unixExcs
hasIsoLocale
root
notRoot
eformat
stdio

RUNNING ALL TESTS
CONFIGURABLE OPTIONS

-singlproc  boolean
-debug  level
  0
  1
  2
  3
-verbose  level
  body (b)
  pass (p)
  skip (s)
  start (t)
  error (e)
  line (l)
-preservecore  level
  0
  1
-limitconstraints boolean
-constraints list
-tmpdir directory
-testdir directory
-file patternList
-notfile patternList
-relateddir patternList
-asidefromdir patternList
-match patternList
-skip patternList
-load script
-loadfile filename
-outfile filename
-errfile filename

CREATING TEST SUITES WITH TCLTEST
COMPATIBILITY
KNOWN ISSUES
KEYWORDS

**NAME**

tcltest - Test harness support code and utilities

**SYNOPSIS**

package require tcltest ?2.3?
tcltest::test name description ?option value ...?
tcltest::test name description ?constraints? body result
tcltest::loadTestedCommands
tcltest::makeDirectory name ?directory?
tcltest::removeDirectory name ?directory?
tcltest::makeFile contents name ?directory?
tcltest::removeFile name ?directory?
tcltest::viewFile name ?directory?
tcltest::cleanupTests ?runningMultipleTests?
tcltest::runAllTests
tcltest::configure
The **tcltest** package provides several utility commands useful in the construction of test suites for code instrumented to be run by evaluation of Tcl commands. Notably the built-in commands of the Tcl library itself are tested by a test suite using the tcltest package.
All the commands provided by the tcltest package are defined in and exported from the ::tcltest namespace, as indicated in the SYNOPSIS above. In the following sections, all commands will be described by their simple names, in the interest of brevity.

The central command of tcltest is test that defines and runs a test. Testing with test involves evaluation of a Tcl script and comparing the result to an expected result, as configured and controlled by a number of options. Several other commands provided by tcltest govern the configuration of test and the collection of many test commands into test suites.

See CREATING TEST SUITES WITH TCLTEST below for an extended example of how to use the commands of tcltest to produce test suites for your Tcl-enabled code.

**COMMANDS**

test name description ?option value ...?  
Defines and possibly runs a test with the name *name* and description *description*. The name and description of a test are used in messages reported by test during the test, as configured by the options of tcltest. The remaining *option value* arguments to test define the test, including the scripts to run, the conditions under which to run them, the expected result, and the means by which the expected and actual results should be compared. See TESTS below for a complete description of the valid options and how they define a test. The test command returns an empty string.

test name description ?constraints? body result  
This form of test is provided to support test suites written for version 1 of the tcltest package, and also a simpler interface for a common usage. It is the same as “test name description -constraints constraints -body body -result result”. All other options to test take their default values. When constraints is omitted, this form of test can be distinguished from the first because all options begin with “-”.
loadTestedCommands
Evaluates in the caller's context the script specified by configure -load or configure -loadfile. Returns the result of that script evaluation, including any error raised by the script. Use this command and the related configuration options to provide the commands to be tested to the interpreter running the test suite.

makeFile contents name ?directory?
Creates a file named name relative to directory directory and write contents to that file using the encoding encoding system. If contents does not end with a newline, a newline will be appended so that the file named name does end with a newline. Because the system encoding is used, this command is only suitable for making text files. The file will be removed by the next evaluation of cleanupTests, unless it is removed by removeFile first. The default value of directory is the directory configure -tmpdir. Returns the full path of the file created. Use this command to create any text file required by a test with contents as needed.

removeFile name ?directory?
Forces the file referenced by name to be removed. This file name should be relative to directory. The default value of directory is the directory configure -tmpdir. Returns an empty string. Use this command to delete files created by makeFile.

makeDirectory name ?directory?
Creates a directory named name relative to directory directory. The directory will be removed by the next evaluation of cleanupTests, unless it is removed by removeDirectory first. The default value of directory is the directory configure -tmpdir. Returns the full path of the directory created. Use this command to create any directories that are required to exist by a test.

removeDirectory name ?directory?
Forces the directory referenced by name to be removed. This directory should be relative to directory. The default value of directory is the directory configure -tmpdir. Returns an empty string. Use this command to delete any directories created by
makeDirectory.

**viewFile** *file ?directory?*

Returns the contents of *file*, except for any final newline, just as `read -nonewline` would return. This file name should be relative to *directory*. The default value of *directory* is the directory `configure -tmpdir`. Use this command as a convenient way to turn the contents of a file generated by a test into the result of that test for matching against an expected result. The contents of the file are read using the system encoding, so its usefulness is limited to text files.

**cleanupTests**

Intended to clean up and summarize after several tests have been run. Typically called once per test file, at the end of the file after all tests have been completed. For best effectiveness, be sure that the `cleanupTests` is evaluated even if an error occurs earlier in the test file evaluation.

Prints statistics about the tests run and removes files that were created by `makeDirectory` and `makeFile` since the last `cleanupTests`. Names of files and directories in the directory `configure -tmpdir` created since the last `cleanupTests`, but not created by `makeFile` or `makeDirectory` are printed to `outputChannel`. This command also restores the original shell environment, as described by the `::env` array. Returns an empty string.

**runAllTests**

This is a master command meant to run an entire suite of tests, spanning multiple files and/or directories, as governed by the configurable options of `tcltest`. See **RUNNING ALL TESTS** below for a complete description of the many variations possible with `runAllTests`.

### CONFIGURATION COMMANDS

**configure**
Returns the list of configurable options supported by `tcltest`. See `CONFIGURABLE OPTIONS` below for the full list of options, their valid values, and their effect on `tcltest` operations.

**configure** option

Returns the current value of the supported configurable option `option`. Raises an error if `option` is not a supported configurable option.

**configure** option value ?option value ...?

Sets the value of each configurable option `option` to the corresponding value `value`, in order. Raises an error if an `option` is not a supported configurable option, or if `value` is not a valid value for the corresponding `option`, or if a `value` is not provided. When an error is raised, the operation of `configure` is halted, and subsequent `option value` arguments are not processed.

If the environment variable `::env(TCLTEST_OPTIONS)` exists when the `tcltest` package is loaded (by `package require tcltest`) then its value is taken as a list of arguments to pass to `configure`. This allows the default values of the configuration options to be set by the environment.

**customMatch** mode script

Registers `mode` as a new legal value of the `-match` option to `test`. When the `-match mode` option is passed to `test`, the script `script` will be evaluated to compare the actual result of evaluating the body of the test to the expected result. To perform the match, the `script` is completed with two additional words, the expected result, and the actual result, and the completed script is evaluated in the global namespace. The completed script is expected to return a boolean value indicating whether or not the results match. The built-in matching modes of `test` are `exact`, `glob`, and `regexp`.

**testConstraint** constraint ?boolean?

Sets or returns the boolean value associated with the named `constraint`. See `TEST CONSTRAINTS` below for more information.
**interpreter** ?executableName?  
Sets or returns the name of the executable to be executed by `runAllTests` to run each test file when `configure -singleproc` is false. The default value for `interpreter` is the name of the currently running program as returned by `info nameofexecutable`.

**outputChannel** ?channelID?  
Sets or returns the output channel ID. This defaults to stdout. Any test that prints test related output should send that output to `outputChannel` rather than letting that output default to stdout.

**errorChannel** ?channelID?  
Sets or returns the error channel ID. This defaults to stderr. Any test that prints error messages should send that output to `errorChannel` rather than printing directly to stderr.

### SHORTCUT COMMANDS

**debug** ?level?  
Same as `configure -debug ?level?`.

**errorFile** ?filename?  
Same as `configure -errfile ?filename?`.

**limitConstraints** ?boolean?  
Same as `configure -limitconstraints ?boolean?`.

**loadFile** ?filename?  
Same as `configure -loadfile ?filename?`.

**loadScript** ?script?  
Same as `configure -load ?script?`.

**match** ?patternList?  
Same as `configure -match ?patternList?`.

**matchDirectories** ?patternList?  
Same as `configure -relateddir ?patternList?`.
matchFiles ?patternList?
   Same as configure -file ?patternList?.

outputFile ?filename?
   Same as configure -outfile ?filename?.

preserveCore ?level?
   Same as configure -preservecore ?level?.

singleProcess ?boolean?
   Same as configure -singleproc ?boolean?.

skip ?patternList?
   Same as configure -skip ?patternList?.

skipDirectories ?patternList?
   Same as configure -asidefromdir ?patternList?.

skipFiles ?patternList?
   Same as configure -notfile ?patternList?.

temporaryDirectory ?directory?
   Same as configure -tmpdir ?directory?.

testsDirectory ?directory?
   Same as configure -testdir ?directory?.

verbose ?level?
   Same as configure -verbose ?level?.

OTHER COMMANDS

The remaining commands provided by tcltest have better alternatives provided by tcltest or Tcl itself. They are retained to support existing test suites, but should be avoided in new code.

test name description optionList
   This form of test was provided to enable passing many options spanning several lines to test as a single argument quoted by
braces, rather than needing to backslash quote the newlines between arguments to `test`. The `optionList` argument is expected to be a list with an even number of elements representing `option` and `value` arguments to pass to `test`. However, these values are not passed directly, as in the alternate forms of `switch`. Instead, this form makes an unfortunate attempt to overthrow Tcl's substitution rules by performing substitutions on some of the list elements as an attempt to implement a “do what I mean” interpretation of a brace-enclosed “block”. The result is nearly impossible to document clearly, and for that reason this form is not recommended. See the examples in `CREATING TEST SUITES WITH TCLTEST` below to see that this form is really not necessary to avoid backslash-quoted newlines. If you insist on using this form, examine the source code of `tcltest` if you want to know the substitution details, or just enclose the third through last argument to `test` in braces and hope for the best.

**workingDirectory ?directoryName?**
Sets or returns the current working directory when the test suite is running. The default value for `workingDirectory` is the directory in which the test suite was launched. The Tcl commands `cd` and `pwd` are sufficient replacements.

**normalizeMsg msg**
Returns the result of removing the “extra” newlines from `msg`, where “extra” is rather imprecise. Tcl offers plenty of string processing commands to modify strings as you wish, and `customMatch` allows flexible matching of actual and expected results.

**normalizePath pathVar**
Resolves symlinks in a path, thus creating a path without internal redirection. It is assumed that `pathVar` is absolute. `pathVar` is modified in place. The Tcl command `file normalize` is a sufficient replacement.

**bytestring string**
Construct a string that consists of the requested sequence of bytes,
as opposed to a string of properly formed UTF-8 characters using the value supplied in string. This allows the tester to create denormalized or improperly formed strings to pass to C procedures that are supposed to accept strings with embedded NULL types and confirm that a string result has a certain pattern of bytes. This is exactly equivalent to the Tcl command encoding convertfrom identity.

### TESTS

The **test** command is the heart of the **tcltest** package. Its essential function is to evaluate a Tcl script and compare the result with an expected result. The options of **test** define the test script, the environment in which to evaluate it, the expected result, and how the compare the actual result to the expected result. Some configuration options of **tcltest** also influence how **test** operates.

The valid options for **test** are summarized:

```
test name description
  ?-constraints keywordList|expression?
  ?-setup setupScript?
  ?-body testScript?
  ?-cleanup cleanupScript?
  ?-result expectedAnswer?
  ?-output expectedOutput?
  ?-errorOutput expectedError?
  ?-returnCodes codeList?
  ?-match mode?
```

The **name** may be any string. It is conventional to choose a **name** according to the pattern:

```
target-majorNum.minorNum
```
For white-box (regression) tests, the target should be the name of the C function or Tcl procedure being tested. For black-box tests, the target should be the name of the feature being tested. Some conventions call for the names of black-box tests to have the suffix _bb. Related tests should share a major number. As a test suite evolves, it is best to have the same test name continue to correspond to the same test, so that it remains meaningful to say things like “Test foo-1.3 passed in all releases up to 3.4, but began failing in release 3.5.”

During evaluation of test, the name will be compared to the lists of string matching patterns returned by configure -match, and configure -skip. The test will be run only if name matches any of the patterns from configure -match and matches none of the patterns from configure -skip.

The description should be a short textual description of the test. The description is included in output produced by the test, typically test failure messages. Good description values should briefly explain the purpose of the test to users of a test suite. The name of a Tcl or C function being tested should be included in the description for regression tests. If the test case exists to reproduce a bug, include the bug ID in the description.

Valid attributes and associated values are:

-constraints keywordList|expression
The optional -constraints attribute can be list of one or more keywords or an expression. If the -constraints value is a list of keywords, each of these keywords should be the name of a constraint defined by a call to testConstraint. If any of the listed constraints is false or does not exist, the test is skipped. If the -constraints value is an expression, that expression is evaluated. If the expression evaluates to true, then the test is run. Note that the expression form of -constraints may interfere with the operation of configure -constraints and configure -limitconstraints, and is not recommended. Appropriate constraints should be added to any tests that should not always be run. That is, conditional evaluation of a test should be accomplished by the -constraints option, not by
conditional evaluation of test. In that way, the same number of tests are always reported by the test suite, though the number skipped may change based on the testing environment. The default value is an empty list. See TEST CONSTRAINTS below for a list of built-in constraints and information on how to add your own constraints.

-**setup** script

The optional -setup attribute indicates a script that will be run before the script indicated by the -body attribute. If evaluation of script raises an error, the test will fail. The default value is an empty script.

-**body** script

The -body attribute indicates the script to run to carry out the test. It must return a result that can be checked for correctness. If evaluation of script raises an error, the test will fail. The default value is an empty script.

-**cleanup** script

The optional -cleanup attribute indicates a script that will be run after the script indicated by the -body attribute. If evaluation of script raises an error, the test will fail. The default value is an empty script.

-**match** mode

The -match attribute determines how expected answers supplied by -result, -output, and -errorOutput are compared. Valid values for mode are regexp, glob, exact, and any value registered by a prior call to customMatch. The default value is exact.

-**result** expectedValue

The -result attribute supplies the expectedValue against which the return value from script will be compared. The default value is an empty string.

-**output** expectedValue

The -output attribute supplies the expectedValue against which
any output sent to stdout or outputChannel during evaluation of the script(s) will be compared. Note that only output printed using \texttt{::puts} is used for comparison. If \texttt{-output} is not specified, output sent to stdout and outputChannel is not processed for comparison.

\textbf{-errorOutput \textit{expectedValue}}

The \texttt{-errorOutput} attribute supplies the \textit{expectedValue} against which any output sent to stderr or errorChannel during evaluation of the script(s) will be compared. Note that only output printed using \texttt{::puts} is used for comparison. If \texttt{-errorOutput} is not specified, output sent to stderr and errorChannel is not processed for comparison.

\textbf{-returnCodes \textit{expectedCodeList}}

The optional \texttt{-returnCodes} attribute supplies \textit{expectedCodeList}, a list of return codes that may be accepted from evaluation of the \texttt{-body} script. If evaluation of the \texttt{-body} script returns a code not in the \textit{expectedCodeList}, the test fails. All return codes known to \texttt{return}, in both numeric and symbolic form, including extended return codes, are acceptable elements in the \textit{expectedCodeList}. Default value is “ok”\texttt{return}.

To pass, a test must successfully evaluate its \texttt{-setup}, \texttt{-body}, and \texttt{-cleanup} scripts. The return code of the \texttt{-body} script and its result must match expected values, and if specified, output and error data from the test must match expected \texttt{-output} and \texttt{-errorOutput} values. If any of these conditions are not met, then the test fails. Note that all scripts are evaluated in the context of the caller of \texttt{test}.

As long as \texttt{test} is called with valid syntax and legal values for all attributes, it will not raise an error. Test failures are instead reported as output written to outputChannel. In default operation, a successful test produces no output. The output messages produced by \texttt{test} are controlled by the \texttt{configure -verbose} option as described in \texttt{CONFIGURABLE OPTIONS} below. Any output produced by the test scripts themselves should be produced using \texttt{::puts} to outputChannel or errorChannel, so that users of the test suite may easily capture
output with the configure -outfile and configure -errfile options, and so that the -output and -errorOutput attributes work properly.

---

TEST CONSTRAINTS

Constraints are used to determine whether or not a test should be skipped. Each constraint has a name, which may be any string, and a boolean value. Each test has a -constraints value which is a list of constraint names. There are two modes of constraint control. Most frequently, the default mode is used, indicated by a setting of configure -limitconstraints to false. The test will run only if all constraints in the list are true-valued. Thus, the -constraints option of test is a convenient, symbolic way to define any conditions required for the test to be possible or meaningful. For example, a test with -constraints unix will only be run if the constraint unix is true, which indicates the test suite is being run on a Unix platform.

Each test should include whatever -constraints are required to constrain it to run only where appropriate. Several constraints are pre-defined in the tcltest package, listed below. The registration of user-defined constraints is performed by the testConstraint command. User-defined constraints may appear within a test file, or within the script specified by the configure -load or configure -loadfile options.

The following is a list of constraints pre-defined by the tcltest package itself:

- **singleTestInterp**: test can only be run if all test files are sourced into a single interpreter
- **unix**: test can only be run on any Unix platform
- **win**: test can only be run on any Windows platform
- **nt**
test can only be run on any Windows NT platform

95
test can only be run on any Windows 95 platform

98
test can only be run on any Windows 98 platform

mac
test can only be run on any Mac platform

unixOrWin
test can only be run on a Unix or Windows platform

macOrWin
test can only be run on a Mac or Windows platform

macOrUnix
test can only be run on a Mac or Unix platform

tempNotWin
test can not be run on Windows. This flag is used to temporarily disable a test.

tempNotMac
test can not be run on a Mac. This flag is used to temporarily disable a test.

unixCrash
test crashes if it is run on Unix. This flag is used to temporarily disable a test.

winCrash
test crashes if it is run on Windows. This flag is used to temporarily disable a test.

macCrash
test crashes if it is run on a Mac. This flag is used to temporarily disable a test.
emptyTest
test is empty, and so not worth running, but it remains as a placeholder for a test to be written in the future. This constraint has value false to cause tests to be skipped unless the user specifies otherwise.

knownBug
test is known to fail and the bug is not yet fixed. This constraint has value false to cause tests to be skipped unless the user specifies otherwise.

nonPortable
test can only be run in some known development environment. Some tests are inherently non-portable because they depend on things like word length, file system configuration, window manager, etc. This constraint has value false to cause tests to be skipped unless the user specifies otherwise.

userInteraction
test requires interaction from the user. This constraint has value false to causes tests to be skipped unless the user specifies otherwise.

interactive
test can only be run in if the interpreter is in interactive mode (when the global tcl_interactive variable is set to 1).

nonBlockFiles
test can only be run if platform supports setting files into nonblocking mode

asyncPipeClose
test can only be run if platform supports async flush and async close on a pipe

unixExecs
test can only be run if this machine has Unix-style commands cat, echo, sh, wc, rm, sleep, fgrep, ps, chmod, and mkdir available
**hasIsoLocale**

- test can only be run if can switch to an ISO locale

**root**

- test can only run if Unix user is root

**notRoot**

- test can only run if Unix user is not root

**eformat**

- test can only run if app has a working version of sprintf with respect to the “e” format of floating-point numbers.

**stdio**

- test can only be run if interpreter can be opened as a pipe.

The alternative mode of constraint control is enabled by setting `configure -limitconstraints` to true. With that configuration setting, all existing constraints other than those in the constraint list returned by `configure -constraints` are set to false. When the value of `configure -constraints` is set, all those constraints are set to true. The effect is that when both options `configure -constraints` and `configure -limitconstraints` are in use, only those tests including only constraints from the `configure -constraints` list are run; all others are skipped. For example, one might set up a configuration with

```
configure -constraints knownBug \ 
-limitconstraints true \ 
-verbose pass
```

to run exactly those tests that exercise known bugs, and discover whether any of them pass, indicating the bug had been fixed.

**RUNNING ALL TESTS**

The single command `runAllTests` is evaluated to run an entire test suite, spanning many files and directories. The configuration options of
tcltest control the precise operations. The runAllTests command begins by printing a summary of its configuration to outputChannel.

Test files to be evaluated are sought in the directory configure -testdir. The list of files in that directory that match any of the patterns in configure -file and match none of the patterns in configure -notfile is generated and sorted. Then each file will be evaluated in turn. If configure -singleproc is true, then each file will be sourced in the caller's context. If it is false, then a copy of interpreter will be exec'd to evaluate each file. The multi-process operation is useful when testing can cause errors so severe that a process terminates. Although such an error may terminate a child process evaluating one file, the master process can continue with the rest of the test suite. In multi-process operation, the configuration of tcltest in the master process is passed to the child processes as command line arguments, with the exception of configure -outfile. The runAllTests command in the master process collects all output from the child processes and collates their results into one master report. Any reports of individual test failures, or messages requested by a configure -verbose setting are passed directly on to outputChannel by the master process.

After evaluating all selected test files, a summary of the results is printed to outputChannel. The summary includes the total number of tests evaluated, broken down into those skipped, those passed, and those failed. The summary also notes the number of files evaluated, and the names of any files with failing tests or errors. A list of the constraints that caused tests to be skipped, and the number of tests skipped for each is also printed. Also, messages are printed if it appears that evaluation of a test file has caused any temporary files to be left behind in configure -tmpdir.

Having completed and summarized all selected test files, runAllTests then recursively acts on subdirectories of configure -testdir. All subdirectories that match any of the patterns in configure -relateddir and do not match any of the patterns in configure -asidefromdir are examined. If a file named all.tcl is found in such a directory, it will be sourced in the caller's context. Whether or not an examined directory contains an all.tcl file, its subdirectories are also scanned against the
configure -relateddir and configure -asidefromdir patterns. In this way, many directories in a directory tree can have all their test files evaluated by a single runAllTests command.

**CONFIGURABLE OPTIONS**

The configure command is used to set and query the configurable options of tcltest. The valid options are:

**-singleproc boolean**
Controls whether or not runAllTests spawns a child process for each test file. No spawning when boolean is true. Default value is false.

**-debug level**
Sets the debug level to level, an integer value indicating how much debugging information should be printed to stdout. Note that debug messages always go to stdout, independent of the value of configure -outfile. Default value is 0. Levels are defined as:

0
Do not display any debug information.

1
Display information regarding whether a test is skipped because it does not match any of the tests that were specified using by configure -match (userSpecifiedNonMatch) or matches any of the tests specified by configure -skip (userSpecifiedSkip). Also print warnings about possible lack of cleanup or balance in test files. Also print warnings about any re-use of test names.

2
Display the flag array parsed by the command line processor, the contents of the ::env array, and all user-defined variables that exist in the current namespace as they are used.

3
Display information regarding what individual procs in the test harness are doing.

-verbose level
Sets the type of output verbosity desired to level, a list of zero or more of the elements body, pass, skip, start, error and line. Default value is {body error}. Levels are defined as:

body (b)
  Display the body of failed tests

pass (p)
  Print output when a test passes

skip (s)
  Print output when a test is skipped

start (t)
  Print output whenever a test starts

error (e)
  Print errorInfo and errorCode, if they exist, when a test return code does not match its expected return code

line (l)
  Print source file line information of failed tests

The single letter abbreviations noted above are also recognized so that “configure -verbose pt” is the same as “configure -verbose {pass start}”.

-preservecore level
Sets the core preservation level to level. This level determines how stringent checks for core files are. Default value is 0. Levels are defined as:

0
  No checking — do not check for core files at the end of each test command, but do check for them in runAllTests after all
test files have been evaluated.

1
   Also check for core files at the end of each test command.

2
   Check for core files at all times described above, and save a copy of each core file produced in configure -tmpdir.

-limitconstraints boolean
   Sets the mode by which test honors constraints as described in TESTS above. Default value is false.

-constraints list
   Sets all the constraints in list to true. Also used in combination with configure -limitconstraints true to control an alternative constraint mode as described in TESTS above. Default value is an empty list.

-tmpdir directory
   Sets the temporary directory to be used by makeFile, makeDirectory, viewFile, removeFile, and removeDirectory as the default directory where temporary files and directories created by test files should be created. Default value is workingDirectory.

-testdir directory
   Sets the directory searched by runAllTests for test files and subdirectories. Default value is workingDirectory.

-file patternList
   Sets the list of patterns used by runAllTests to determine what test files to evaluate. Default value is “*.test”.

-notfile patternList
   Sets the list of patterns used by runAllTests to determine what test files to skip. Default value is “I.*.test”, so that any SCCS lock files are skipped.

-relateddir patternList
Sets the list of patterns used by `runAllTests` to determine what subdirectories to search for an `all.tcl` file. Default value is “*”.

**-asidefromdir** `patternList`
Sets the list of patterns used by `runAllTests` to determine what subdirectories to skip when searching for an `all.tcl` file. Default value is an empty list.

**-match** `patternList`
Set the list of patterns used by `test` to determine whether a test should be run. Default value is “*”.

**-skip** `patternList`
Set the list of patterns used by `test` to determine whether a test should be skipped. Default value is an empty list.

**-load** `script`
Sets a script to be evaluated by `loadTestedCommands`. Default value is an empty script.

**-loadfile** `filename`
Sets the filename from which to read a script to be evaluated by `loadTestedCommands`. This is an alternative to `-load`. They cannot be used together.

**-outfile** `filename`
Sets the file to which all output produced by tcltest should be written. A file named `filename` will be opened for writing, and the resulting channel will be set as the value of `outputChannel`.

**-errfile** `filename`
Sets the file to which all error output produced by tcltest should be written. A file named `filename` will be opened for writing, and the resulting channel will be set as the value of `errorChannel`.

---

**CREATING TEST SUITES WITH TCLTEST**

The fundamental element of a test suite is the individual test command.
We begin with several examples.

[1] Test of a script that returns normally.

```plaintext
test example-1.0 {normal return} {
    format %s value
} value
```

[2] Test of a script that requires context setup and cleanup. Note the bracing and indenting style that avoids any need for line continuation.

```plaintext
test example-1.1 {test file existence} -setup {
    set file [makeFile {} test]
} -body {
    file exists $file
} -cleanup {
    removeFile test
} -result 1
```


```plaintext
test example-1.2 {error return} -body {
    error message
} -returnCodes error -result message
```

At the next higher layer of organization, several test commands are gathered together into a single test file. Test files should have names with the .test extension, because that is the default pattern used by runAllTests to find test files. It is a good rule of thumb to have one test file for each source code file of your project. It is good practice to edit the test file and the source code file together, keeping tests synchronized with code changes.

Most of the code in the test file should be the test commands. Use constraints to skip tests, rather than conditional evaluation of test.

[5] Recommended system for writing conditional tests, using constraints to guard:

```tcl
testConstraint X [expr $myRequirement]
test goodConditionalTest {} X {
    # body
} result
```

[6] Discouraged system for writing conditional tests, using if to guard:
Use the -setup and -cleanup options to establish and release all context requirements of the test body. Do not make tests depend on prior tests in the file. Those prior tests might be skipped. If several consecutive tests require the same context, the appropriate setup and cleanup scripts may be stored in variable for passing to each tests -setup and -cleanup options. This is a better solution than performing setup outside of test commands, because the setup will only be done if necessary, and any errors during setup will be reported, and not cause the test file to abort.

A test file should be able to be combined with other test files and not interfere with them, even when configure -singleproc 1 causes all files to be evaluated in a common interpreter. A simple way to achieve this is to have your tests define all their commands and variables in a namespace that is deleted when the test file evaluation is complete. A good namespace to use is a child namespace test of the namespace of the module you are testing.

A test file should also be able to be evaluated directly as a script, not depending on being called by a master runAllTests. This means that each test file should process command line arguments to give the tester all the configuration control that tcltest provides.

After all tests in a test file, the command cleanupTests should be called.

[7] Here is a sketch of a sample test file illustrating those points:

```tcl
if $myRequirement {
    test badConditionalTest {} {
        #body
    } result
}
```
package require tcltest 2.2
eval ::tcltest::configure $argv
package require example
namespace eval ::example::test {
    namespace import ::tcltest::*
    testConstraint X [expr {...}]
    variable SETUP {#common setup code}
    variable CLEANUP {#common cleanup code}
    test example-1 {} -setup $SETUP -body {
        # First test
    } -cleanup $CLEANUP -result {...}
    test example-2 {} -constraints X -setup $SETUP -body {
        # Second test; constrained
    } -cleanup $CLEANUP -result {...}
    test example-3 {} {
        # Third test; no context required
    } {...}
    cleanupTests
}
namespace delete ::example::test

The next level of organization is a full test suite, made up of several test files. One script is used to control the entire suite. The basic function of this script is to call runAllTests after doing any necessary setup. This script is usually named all.tcl because that is the default name used by runAllTests when combining multiple test suites into one testing run.

[8]
Here is a sketch of a sample test suite master script:

```tcl
package require Tcl 8.4
package require tcltest 2.2
package require example
::tcltest::configure -testdir \
```
COMPATIBILITY

A number of commands and variables in the ::tcltest namespace provided by earlier releases of tcltest have not been documented here. They are no longer part of the supported public interface of tcltest and should not be used in new test suites. However, to continue to support existing test suites written to the older interface specifications, many of those deprecated commands and variables still work as before. For example, in many circumstances, configure will be automatically called shortly after package require tcltest 2.1 succeeds with arguments from the variable ::argv. This is to support test suites that depend on the old behavior that tcltest was automatically configured from command line arguments. New test files should not depend on this, but should explicitly include

```tcl
eval ::tcltest::configure $::argv
```

to establish a configuration from command line arguments.

KNOWN ISSUES

There are two known issues related to nested evaluations of test. The first issue relates to the stack level in which test scripts are executed. Tests nested within other tests may be executed at the same stack level as the outermost test. For example, in the following code:

```tcl
set level 1

test level-1.1 {level 1} {
    -body {
        test level-2.1 {level 2} {
    }
}
```
any script executed in level-2.1 may be executed at the same stack level as the script defined for level-1.1.

In addition, while two tests have been run, results will only be reported by cleanupTests for tests at the same level as test level-1.1. However, test results for all tests run prior to level-1.1 will be available when test level-2.1 runs. What this means is that if you try to access the test results for test level-2.1, it will may say that “m” tests have run, “n” tests have been skipped, “o” tests have passed and “p” tests have failed, where “m”, “n”, “o”, and “p” refer to tests that were run at the same test level as test level-1.1.

Implementation of output and error comparison in the test command depends on usage of ::=puts in your application code. Output is intercepted by redefining the ::=puts command while the defined test script is being run. Errors thrown by C procedures or printed directly from C applications will not be caught by the test command. Therefore, usage of the -output and -errorOutput options to test is useful only for pure Tcl applications that use ::=puts to produce output.

**KEYWORDS**

test, test harness, test suite
NAME
fconfigure - Set and get options on a channel

SYNOPSIS

DESCRIPTION
-blocking boolean
-buffering newValue
-buffersize newSize
-encoding name
-eofchar char
-eofchar {inChar outChar}
-translation mode
-translation {inMode outMode}
  auto
  binary
  cr
  crlf
  lf

STANDARD CHANNELS

EXAMPLES

SEE ALSO

KEYWORDS

NAME
fconfigure - Set and get options on a channel

SYNOPSIS

fconfigure channelld
fconfigure channelld name
fconfigure channelld name value ?name value ...?
**DESCRIPTION**

The **fconfigure** command sets and retrieves options for channels.

*ChannelId* identifies the channel for which to set or query an option and must refer to an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of **open** or **socket**, or the result of a channel creation command provided by a Tcl extension.

If no *name* or *value* arguments are supplied, the command returns a list containing alternating option names and values for the channel. If *name* is supplied but no *value* then the command returns the current value of the given option. If one or more pairs of *name* and *value* are supplied, the command sets each of the named options to the corresponding *value*; in this case the return value is an empty string.

The options described below are supported for all channels. In addition, each channel type may add options that only it supports. See the manual entry for the command that creates each type of channel for the options that that specific type of channel supports. For example, see the manual entry for the **socket** command for its additional options.

**-blocking boolean**

The **-blocking** option determines whether I/O operations on the channel can cause the process to block indefinitely. The value of the option must be a proper boolean value. Channels are normally in blocking mode; if a channel is placed into nonblocking mode it will affect the operation of the **gets**, **read**, **puts**, **flush**, and **close** commands by allowing them to operate asynchronously; see the documentation for those commands for details. For nonblocking mode to work correctly, the application must be using the Tcl event loop (e.g. by calling **Tcl_DoOneEvent** or invoking the **vwait** command).

**-buffering newValue**

If *newValue* is **full** then the I/O system will buffer output until its internal buffer is full or until the **flush** command is invoked. If
newValue is line, then the I/O system will automatically flush output for the channel whenever a newline character is output. If newValue is none, the I/O system will flush automatically after every output operation. The default is for -buffering to be set to full except for channels that connect to terminal-like devices; for these channels the initial setting is line. Additionally, stdin and stdout are initially set to line, and stderr is set to none.

-buffersize newSize
Newvalue must be an integer; its value is used to set the size of buffers, in bytes, subsequently allocated for this channel to store input or output. Newvalue must be between ten and one million, allowing buffers of ten to one million bytes in size.

-encoding name
This option is used to specify the encoding of the channel, so that the data can be converted to and from Unicode for use in Tcl. For instance, in order for Tcl to read characters from a Japanese file in shiftjis and properly process and display the contents, the encoding would be set to shiftjis. Thereafter, when reading from the channel, the bytes in the Japanese file would be converted to Unicode as they are read. Writing is also supported - as Tcl strings are written to the channel they will automatically be converted to the specified encoding on output.

If a file contains pure binary data (for instance, a JPEG image), the encoding for the channel should be configured to be binary. Tcl will then assign no interpretation to the data in the file and simply read or write raw bytes. The Tcl binary command can be used to manipulate this byte-oriented data. It is usually better to set the -translation option to binary when you want to transfer binary data, as this turns off the other automatic interpretations of the bytes in the stream as well.

The default encoding for newly opened channels is the same platform- and locale-dependent system encoding used for interfacing with the operating system, as returned by encoding system.
**-eofchar char**

This option supports DOS file systems that use Control-z (\x1a) as an end of file marker. If char is not an empty string, then this character signals end-of-file when it is encountered during input. For output, the end-of-file character is output when the channel is closed. If char is the empty string, then there is no special end of file character marker. For read-write channels, a two-element list specifies the end of file marker for input and output, respectively. As a convenience, when setting the end-of-file character for a read-write channel you can specify a single value that will apply to both reading and writing. When querying the end-of-file character of a read-write channel, a two-element list will always be returned. The default value for -eofchar is the empty string in all cases except for files under Windows. In that case the -eofchar is Control-z (\x1a) for reading and the empty string for writing. The acceptable range for -eofchar values is \x01 - \x7f; attempting to set -eofchar to a value outside of this range will generate an error.

**-translation mode**

**-translation {inMode outMode}**

In Tcl scripts the end of a line is always represented using a single newline character (\n). However, in actual files and devices the end of a line may be represented differently on different platforms, or even for different devices on the same platform. For example, under UNIX newlines are used in files, whereas carriage-return-linefeed sequences are normally used in network connections. On input (i.e., with gets and read) the Tcl I/O system automatically translates the external end-of-line representation into newline characters. Upon output (i.e., with puts), the I/O system translates newlines to the external end-of-line representation. The default translation mode, auto, handles all the common cases automatically, but the -translation option provides explicit control over the end of line translations.

The value associated with -translation is a single item for read-
only and write-only channels. The value is a two-element list for read-write channels; the read translation mode is the first element of the list, and the write translation mode is the second element. As a convenience, when setting the translation mode for a read-write channel you can specify a single value that will apply to both reading and writing. When querying the translation mode of a read-write channel, a two-element list will always be returned. The following values are currently supported:

**auto**

As the input translation mode, **auto** treats any of newline (**lf**), carriage return (**cr**), or carriage return followed by a newline (**crlf**) as the end of line representation. The end of line representation can even change from line-to-line, and all cases are translated to a newline. As the output translation mode, **auto** chooses a platform specific representation; for sockets on all platforms Tcl chooses **crlf**, for all Unix flavors, it chooses **lf**, and for the various flavors of Windows it chooses **crlf**. The default setting for **-translation** is **auto** for both input and output.

**binary**

No end-of-line translations are performed. This is nearly identical to **lf** mode, except that in addition **binary** mode also sets the end-of-file character to the empty string (which disables it) and sets the encoding to **binary** (which disables encoding filtering). See the description of **-eofchar** and **-encoding** for more information.

Internally, i.e. when it comes to the actual behaviour of the translator this value is identical to **lf** and is therefore reported as such when queried. Even if **binary** was used to set the translation.

**cr**

The end of a line in the underlying file or device is represented by a single carriage return character. As the input translation mode, **cr** mode converts carriage returns to newline
characters. As the output translation mode, cr mode translates newline characters to carriage returns.

**crlf**

The end of a line in the underlying file or device is represented by a carriage return character followed by a linefeed character. As the input translation mode, crlf mode converts carriage-return-linefeed sequences to newline characters. As the output translation mode, crlf mode translates newline characters to carriage-return-linefeed sequences. This mode is typically used on Windows platforms and for network connections.

**lf**

The end of a line in the underlying file or device is represented by a single newline (linefeed) character. In this mode no translations occur during either input or output. This mode is typically used on UNIX platforms.

### STANDARD CHANNELS

The Tcl standard channels (stdin, stdout, and stderr) can be configured through this command like every other channel opened by the Tcl library. Beyond the standard options described above they will also support any special option according to their current type. If, for example, a Tcl application is started by the inet super-server common on Unix system its Tcl standard channels will be sockets and thus support the socket options.

### EXAMPLES

Instruct Tcl to always send output to stdout immediately, whether or not it is to a terminal:

```
    fconfigure stdout -buffering none
```

Open a socket and read lines from it without ever blocking the
processing of other events:

```tcl
set s [socket some.where.com 12345]
фconfigure $s -blocking 0
fileevent $s readable "readMe $s"
proc readMe chan {
    if {[gets $chan line] < 0} {
        if {[eof $chan]} {
            close $chan
            return
        }
        # Could not read a complete line this time; Tcl's
        # internal buffering will hold the partial line
        # until some more data is available over the socket.
    } else {
        puts stdout $line
    }
}
```

Read a PPM-format image from a file:

```tcl
# Open the file and put it into Unix ASCII mode
set f [open teapot.ppm]
фconfigure $f -encoding ascii -translation lf

# Get the header
if {[gets $f] ne "P6"} {
    error "not a raw-bits PPM"
}

# Read lines until we have got non-comment lines
# that supply us with three decimal values.
set words {}
while {[llength $words] < 3} {
```
gets $f line
if {[string match "\#*" $line]} continue
lappend words {*[join [scan $line %d%d%d]]}

# Those words supply the size of the image and its
# overall depth per channel. Assign to variables.
lassign $words xSize ySize depth

# Now switch to binary mode to pull in the data,
# one byte per channel (red,green,blue) per pixel.
\textbf{fconfigure} $f -translation binary
set numDataBytes [expr {3 * $xSize * $ySize}]
set data [read $f $numDataBytes]
close $f

\textbf{SEE ALSO}
\texttt{close}, \texttt{flush}, \texttt{gets}, \texttt{open}, \texttt{puts}, \texttt{read}, \texttt{socket}, \texttt{Tcl\_StandardChannels}

\textbf{KEYWORDS}
\texttt{blocking}, \texttt{buffering}, \texttt{carriage\ return}, \texttt{end\ of\ line}, \texttt{flushing}, \texttt{linemode},
\texttt{newline}, \texttt{nonblocking}, \texttt{platform}, \texttt{translation}, \texttt{encoding}, \texttt{filter}, \texttt{byte\ array},
\texttt{binary}
NAME

load - Load machine code and initialize new commands

SYNOPSIS

load fileName
load fileName packageName
load fileName packageName interp

DESCRIPTION

This command loads binary code from a file into the application's address space and calls an initialization procedure in the package to incorporate it into an interpreter. fileName is the name of the file containing the code; its exact form varies from system to system but on most systems it is a shared library, such as a .so file under Solaris or a DLL under Windows. packageName is the name of the package, and is used to compute the name of an initialization procedure. interp is the path name of the interpreter into which to load the package (see the interp manual entry for details); if interp is omitted, it defaults to the interpreter in which the load command was invoked.

Once the file has been loaded into the application's address space, one of two initialization procedures will be invoked in the new code. Typically the initialization procedure will add new commands to a Tcl interpreter. The name of the initialization procedure is determined by packageName and whether or not the target interpreter is a safe one. For normal interpreters the name of the initialization procedure will have the form pkg_Init, where pkg is the same as packageName except that the first letter is converted to upper case and all other letters are converted to lower case. For example, if packageName is foo or FOo,
the initialization procedure's name will be **Foo_Init**.

If the target interpreter is a safe interpreter, then the name of the initialization procedure will be `pkg_SafeInit` instead of `pkg_Init`. The `pkg_SafeInit` function should be written carefully, so that it initializes the safe interpreter only with partial functionality provided by the package that is safe for use by untrusted code. For more information on Safe-Tcl, see the **safe** manual entry.

The initialization procedure must match the following prototype:

```c
typedef int Tcl Package_InitProc(Tcl_Interp *interp);
```

The `interp` argument identifies the interpreter in which the package is to be loaded. The initialization procedure must return `TCL_OK` or `TCL_ERROR` to indicate whether or not it completed successfully; in the event of an error it should set the interpreter's result to point to an error message. The result of the `load` command will be the result returned by the initialization procedure.

The actual loading of a file will only be done once for each `fileName` in an application. If a given `fileName` is loaded into multiple interpreters, then the first `load` will load the code and call the initialization procedure; subsequent `loads` will call the initialization procedure without loading the code again. For Tcl versions lower than 8.5, it is not possible to unload or reload a package. From version 8.5 however, the `unload` command allows the unloading of libraries loaded with `load`, for libraries that are aware of the Tcl's unloading mechanism.

The `load` command also supports packages that are statically linked with the application, if those packages have been registered by calling the `Tcl_StaticPackage` procedure. If `fileName` is an empty string, then `packageName` must be specified.

If `packageName` is omitted or specified as an empty string, Tcl tries to guess the name of the package. This may be done differently on
different platforms. The default guess, which is used on most UNIX platforms, is to take the last element of `fileName`, strip off the first three characters if they are `lib`, and use any following alphabetic and underline characters as the module name. For example, the command `load libxyz4.2.so` uses the module name `xyz` and the command `load bin/last.so {}` uses the module name `last`.

If `fileName` is an empty string, then `packageName` must be specified. The `load` command first searches for a statically loaded package (one that has been registered by calling the `Tcl_StaticPackage` procedure) by that name; if one is found, it is used. Otherwise, the `load` command searches for a dynamically loaded package by that name, and uses it if it is found. If several different files have been loaded with different versions of the package, Tcl picks the file that was loaded first.

### PORTABILITY ISSUES

**Windows**

When a load fails with “library not found” error, it is also possible that a dependent library was not found. To see the dependent libraries, type “`dumpbin -imports <dllname>`” in a DOS console to see what the library must import. When loading a DLL in the current directory, Windows will ignore “./” as a path specifier and use a search heuristic to find the DLL instead. To avoid this, load the DLL with:

```
load [file join [pwd] mylib.DLL]
```

### BUGS

If the same file is loaded by different `fileNames`, it will be loaded into the process’s address space multiple times. The behavior of this varies from system to system (some systems may detect the redundant loads, others may not).

### EXAMPLE
The following is a minimal extension:

```c
#include <tcl.h>
#include <stdio.h>

static int fooCmd(ClientData clientData, Tcl_Interp *interp, int objc, Tcl_Obj *const objv[])
{
    printf("called with %d arguments\n", objc);
    return TCL_OK;
}

int Foo_Init(Tcl_Interp *interp) {
    if (Tcl_InitStubs(interp, "8.1", 0) == NULL) {
        return TCL_ERROR;
    }
    printf("creating foo command");
    Tcl_CreateObjCommand(interp, "foo", fooCmd, NULL, NULL);
    return TCL_OK;
}
```

When built into a shared/dynamic library with a suitable name (e.g. `foo.dll` on Windows, `libfoo.so` on Solaris and Linux) it can then be loaded into Tcl with the following:

```c
# Load the extension
switch $tcl_platform(platform) {
    windows {
        load [file join [pwd] foo.dll]
    }
    unix {
        load [file join [pwd] libfoo[$info sharedlibext]
    }
}

# Now execute the command defined by the extension
foo
```
SEE ALSO

info_sharedlibextension, Tcl_StaticPackage, safe

KEYWORDS

binary code, loading, safe interpreter, shared library
NAME

read - Read from a channel

SYNOPSIS

read ?-nonewline? channelId
read channelId numChars

DESCRIPTION

In the first form, the `read` command reads all of the data from `channelId` up to the end of the file. If the `-nonewline` switch is specified then the last character of the file is discarded if it is a newline. In the second form, the extra argument specifies how many characters to read. Exactly that many characters will be read and returned, unless there are fewer than `numChars` left in the file; in this case all the remaining characters are returned. If the channel is configured to use a multi-byte encoding, then the number of characters read may not be the same as the number of bytes read.

`ChannelId` must be an identifier for an open channel such as the Tcl standard input channel (`stdin`), the return value from an invocation of `open` or `socket`, or the result of a channel creation command provided by a Tcl extension. The channel must have been opened for input.

If `channelId` is in nonblocking mode, the command may not read as many characters as requested: once all available input has been read, the command will return the data that is available rather than blocking for more input. If the channel is configured to use a multi-byte encoding, then there may actually be some bytes remaining in the internal buffers that do not form a complete character. These bytes will not be returned
until a complete character is available or end-of-file is reached. The **-nonewline** switch is ignored if the command returns before reaching the end of the file.

**Read** translates end-of-line sequences in the input into newline characters according to the **-translation** option for the channel. See the **fconfigure** manual entry for a discussion on ways in which **fconfigure** will alter input.

**USE WITH SERIAL PORTS**

For most applications a channel connected to a serial port should be configured to be nonblocking: **fconfigure channelId -blocking 0**. Then **read** behaves much like described above. Care must be taken when using **read** on blocking serial ports:

**read channelId numChars**

In this form **read** blocks until **numChars** have been received from the serial port.

**read channelId**

In this form **read** blocks until the reception of the end-of-file character, see **fconfigure -eofchar**. If there no end-of-file character has been configured for the channel, then **read** will block forever.

**EXAMPLE**

This example code reads a file all at once, and splits it into a list, with each line in the file corresponding to an element in the list:

```
set fl [open /proc/meminfo]
set data [read $fl]
close $fl
set lines [split $data 
]
```
file, eof, fblocked, fconfigure, Tcl_StandardChannels

KEYWORDS

blocking, channel, end of line, end of file, nonblocking, read, translation, encoding
NAME
tclvars - Variables used by Tcl

DESCRIPTION
env
errorCode
  ARITH code msg
  CHILDKILLED pid sigName msg
  CHILDSTATUS pid code
  CHILDSUSP pid sigName msg
  NONE
  POSIX errName msg
errorInfo
tcl_library
tcl_patchLevel
tcl_pkgPath
tcl_platform
  byteOrder
debug
machine
os
osVersion
platform
threaded
user
wordSize
pointerSize
tcl_precision
tcl_rcFileName
tcl_traceCompile
tcl_traceExec
tcl_wordchars
tcl_nonwordchars
tcl_version

OTHER GLOBAL VARIABLES
argc
argv
argv0
tcl_interactive
gometry

SEE ALSO
KEYWORDS

NAME
tclvars - Variables used by Tcl

DESCRIPTION

The following global variables are created and managed automatically by the Tcl library. Except where noted below, these variables should normally be treated as read-only by application-specific code and by users.

env

This variable is maintained by Tcl as an array whose elements are the environment variables for the process. Reading an element will return the value of the corresponding environment variable. Setting an element of the array will modify the corresponding environment variable or create a new one if it does not already exist. Unsetting an element of env will remove the corresponding environment variable. Changes to the env array will affect the environment passed to children by commands like exec. If the entire env array is unset then Tcl will stop monitoring env accesses and will not update environment variables.

Under Windows, the environment variables PATH and COMSPEC in any capitalization are converted automatically to upper case. For instance, the PATH variable could be exported by the operating system as “path”, “Path”, “PaTh”, etc., causing otherwise simple Tc
code to have to support many special cases. All other environment variables inherited by Tcl are left unmodified. Setting an env array variable to blank is the same as unsetting it as this is the behavior of the underlying Windows OS. It should be noted that relying on an existing and empty environment variable will not work on Windows and is discouraged for cross-platform usage.

**errorCode**
This variable holds the value of the `-errorcode` return option set by the most recent error that occurred in this interpreter. This list value represents additional information about the error in a form that is easy to process with programs. The first element of the list identifies a general class of errors, and determines the format of the rest of the list. The following formats for `-errorcode` return options are used by the Tcl core; individual applications may define additional formats.

**ARITH code msg**
This format is used when an arithmetic error occurs (e.g. an attempt to divide zero by zero in the `expr` command). `Code` identifies the precise error and `msg` provides a human-readable description of the error. `Code` will be either DIVZERO (for an attempt to divide by zero), DOMAIN (if an argument is outside the domain of a function, such as `acos(-3)`), IOVERFLOW (for integer overflow), OVERFLOW (for a floating-point overflow), or UNKNOWN (if the cause of the error cannot be determined).

Detection of these errors depends in part on the underlying hardware and system libraries.

**CHILDKILLED pid sigName msg**
This format is used when a child process has been killed because of a signal. The `pid` element will be the process's identifier (in decimal). The `sigName` element will be the symbolic name of the signal that caused the process to terminate; it will be one of the names from the include file `signal.h`, such as `SIGPIPE`. The `msg` element will be a short
human-readable message describing the signal, such as “write on pipe with no readers” for **SIGPIPE**.

**CHILDSTATUS** *pid code*
This format is used when a child process has exited with a non-zero exit status. The *pid* element will be the process's identifier (in decimal) and the *code* element will be the exit code returned by the process (also in decimal).

**CHILDSUSP** *pid sigName msg*
This format is used when a child process has been suspended because of a signal. The *pid* element will be the process's identifier, in decimal. The *sigName* element will be the symbolic name of the signal that caused the process to suspend; this will be one of the names from the include file signal.h, such as **SIGTTIN**. The *msg* element will be a short human-readable message describing the signal, such as “background tty read” for **SIGTTIN**.

**NONE**
This format is used for errors where no additional information is available for an error besides the message returned with the error. In these cases the **-errorcode** return option will consist of a list containing a single element whose contents are **NONE**.

**POSIX** *errName msg*
If the first element is **POSIX**, then the error occurred during a POSIX kernel call. The *errName* element will contain the symbolic name of the error that occurred, such as **ENOENT**; this will be one of the values defined in the include file errno.h. The *msg* element will be a human-readable message corresponding to *errName*, such as “no such file or directory” for the **ENOENT** case.

To set the **-errorcode** return option, applications should use library procedures such as **Tcl_SetObjErrorCode**, **Tcl_SetReturnOptions**, and **Tcl_PosixError**, or they may invoke
the -errorcode option of the return command. If none of these methods for setting the error code has been used, the Tcl interpreter will reset the variable to NONE after the next error.

**errorInfo**
This variable holds the value of the -errorinfo return option set by the most recent error that occurred in this interpreter. This string value will contain one or more lines identifying the Tcl commands and procedures that were being executed when the most recent error occurred. Its contents take the form of a stack trace showing the various nested Tcl commands that had been invoked at the time of the error.

**tcl_library**
This variable holds the name of a directory containing the system library of Tcl scripts, such as those used for auto-loading. The value of this variable is returned by the info library command. See the library manual entry for details of the facilities provided by the Tcl script library. Normally each application or package will have its own application-specific script library in addition to the Tcl script library; each application should set a global variable with a name like $app_library (where app is the application's name) to hold the network file name for that application's library directory. The initial value of tcl_library is set when an interpreter is created by searching several different directories until one is found that contains an appropriate Tcl startup script. If the TCL_LIBRARY environment variable exists, then the directory it names is checked first. If TCL_LIBRARY is not set or doesn't refer to an appropriate directory, then Tcl checks several other directories based on a compiled-in default location, the location of the binary containing the application, and the current working directory.

**tcl_patchLevel**
When an interpreter is created Tcl initializes this variable to hold a string giving the current patch level for Tcl, such as 8.4.16 for Tcl 8.4 with the first sixteen official patches, or 8.5b3 for the third beta release of Tcl 8.5. The value of this variable is returned by the info patchlevel command.
tcl_pkgPath
This variable holds a list of directories indicating where packages are normally installed. It is not used on Windows. It typically contains either one or two entries; if it contains two entries, the first is normally a directory for platform-dependent packages (e.g., shared library binaries) and the second is normally a directory for platform-independent packages (e.g., script files). Typically a package is installed as a subdirectory of one of the entries in $tcl_pkgPath. The directories in $tcl_pkgPath are included by default in the auto_path variable, so they and their immediate subdirectories are automatically searched for packages during package require commands. Note: tcl_pkgPath is not intended to be modified by the application. Its value is added to auto_path at startup; changes to tcl_pkgPath are not reflected in auto_path. If you want Tcl to search additional directories for packages you should add the names of those directories to auto_path, not tcl_pkgPath.

tcl_platform
This is an associative array whose elements contain information about the platform on which the application is running, such as the name of the operating system, its current release number, and the machine's instruction set. The elements listed below will always be defined, but they may have empty strings as values if Tcl could not retrieve any relevant information. In addition, extensions and applications may add additional values to the array. The predefined elements are:

byteOrder
The native byte order of this machine: either littleEndian or bigEndian.

debug
If this variable exists, then the interpreter was compiled with and linked to a debug-enabled C run-time. This variable will only exist on Windows, so extension writers can specify which package to load depending on the C run-time library that is in
use. This is not an indication that this core contains symbols.

machine
The instruction set executed by this machine, such as *intel*, *PPC*, *68k*, or *sun4m*. On UNIX machines, this is the value returned by *uname -m*.

os
The name of the operating system running on this machine, such as *Windows 95*, *Windows NT*, or *SunOS*. On UNIX machines, this is the value returned by *uname -s*. On Windows 95 and Windows 98, the value returned will be *Windows 95* to provide better backwards compatibility to Windows 95; to distinguish between the two, check the *osVersion*.

osVersion
The version number for the operating system running on this machine. On UNIX machines, this is the value returned by *uname -r*. On Windows 95, the version will be 4.0; on Windows 98, the version will be 4.10.

platform
Either *windows*, or *unix*. This identifies the general operating environment of the machine.

threaded
If this variable exists, then the interpreter was compiled with threads enabled.

user
This identifies the current user based on the login information available on the platform. This comes from the USER or LOGNAME environment variable on Unix, and the value from GetUserUserName on Windows.

wordSize
This gives the size of the native-machine word in bytes (strictly, it is same as the result of evaluating *sizeof(long)* in C.)
pointerSize
This gives the size of the native-machine pointer in bytes (strictly, it is same as the result of evaluating sizeof(void*) in C.)

tcl_precision
This variable controls the number of digits to generate when converting floating-point values to strings. It defaults to 0. *Applications should not change this value*; it is provided for compatibility with legacy code.

The default value of 0 is special, meaning that Tcl should convert numbers using as few digits as possible while still distinguishing any floating point number from its nearest neighbours. It differs from using an arbitrarily high value for *tcl_precision* in that an inexact number like 1.4 will convert as 1.4 rather than 1.3999999999999999 even though the latter is nearer to the exact value of the binary number.

17 digits is “perfect” for IEEE floating-point in that it allows double-precision values to be converted to strings and back to binary with no loss of information. However, using 17 digits prevents any rounding, which produces longer, less intuitive results. For example, `expr {1.4}` returns 1.3999999999999999 with *tcl_precision* set to 17, vs. 1.4 if *tcl_precision* is 12.

All interpreters in a thread share a single *tcl_precision* value: changing it in one interpreter will affect all other interpreters as well. However, safe interpreters are not allowed to modify the variable.

tcl_rcFileName
This variable is used during initialization to indicate the name of a user-specific startup file. If it is set by application-specific initialization, then the Tcl startup code will check for the existence of this file and *source* it if it exists. For example, for *wish* the variable is set to ~/.wishrc for Unix and ~/wishrc.tcl for Windows.

tcl_traceCompile
The value of this variable can be set to control how much tracing information is displayed during bytecode compilation. By default, tcl_traceCompile is zero and no information is displayed. Setting tcl_traceCompile to 1 generates a one-line summary in stdout whenever a procedure or top-level command is compiled. Setting it to 2 generates a detailed listing in stdout of the bytecode instructions emitted during every compilation. This variable is useful in tracking down suspected problems with the Tcl compiler.

This variable and functionality only exist if TCL_COMPILE_DEBUG was defined during Tcl's compilation.

tcl_traceExec
The value of this variable can be set to control how much tracing information is displayed during bytecode execution. By default, tcl_traceExec is zero and no information is displayed. Setting tcl_traceExec to 1 generates a one-line trace in stdout on each call to a Tcl procedure. Setting it to 2 generates a line of output whenever any Tcl command is invoked that contains the name of the command and its arguments. Setting it to 3 produces a detailed trace showing the result of executing each bytecode instruction. Note that when tcl_traceExec is 2 or 3, commands such as set and incr that have been entirely replaced by a sequence of bytecode instructions are not shown. Setting this variable is useful in tracking down suspected problems with the bytecode compiler and interpreter.

This variable and functionality only exist if TCL_COMPILE_DEBUG was defined during Tcl's compilation.

tcl_wordchars
The value of this variable is a regular expression that can be set to control what are considered “word” characters, for instances like selecting a word by double-clicking in text in Tk. It is platform dependent. On Windows, it defaults to \S, meaning anything but a Unicode space character. Otherwise it defaults to \w, which is any Unicode word character (number, letter, or underscore).
tcl_nonwordchars
The value of this variable is a regular expression that can be set to control what are considered “non-word” characters, for instances like selecting a word by double-clicking in text in Tk. It is platform dependent. On Windows, it defaults to `\s`, meaning any Unicode space character. Otherwise it defaults to `\W`, which is anything but a Unicode word character (number, letter, or underscore).

tcl_version
When an interpreter is created Tcl initializes this variable to hold the version number for this version of Tcl in the form `x.y`. Changes to `x` represent major changes with probable incompatibilities and changes to `y` represent small enhancements and bug fixes that retain backward compatibility. The value of this variable is returned by the `info tclversion` command.

OTHER GLOBAL VARIABLES

The following variables are only guaranteed to exist in `tclsh` and `wish` executables; the Tcl library does not define them itself but many Tcl environments do.

argc
The number of arguments to `tclsh` or `wish`.

argv
Tcl list of arguments to `tclsh` or `wish`.

argv0
The script that `tclsh` or `wish` started executing (if it was specified) or otherwise the name by which `tclsh` or `wish` was invoked.

tcl_interactive
Contains 1 if `tclsh` or `wish` is running interactively (no script was specified and standard input is a terminal-like device), 0 otherwise.

The `wish` executable additionally specifies the following global variable:
geometry

If set, contains the user-supplied geometry specification to use for the main Tk window.

SEE ALSO

eval, tclsh, wish

KEYWORDS

arithmetic, bytecode, compiler, error, environment, POSIX, precision, subprocess, variables

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME

fcopy - Copy data from one channel to another

SYNOPSIS

fcopy inchan outchan ?-size size? ?-command callback?

DESCRIPTION

The `fcopy` command copies data from one I/O channel, `inchan` to another I/O channel, `outchan`. The `fcopy` command leverages the buffering in the Tcl I/O system to avoid extra copies and to avoid buffering too much data in main memory when copying large files to slow destinations like network sockets.

The `fcopy` command transfers data from `inchan` until end of file or `size` bytes have been transferred. If no `-size` argument is given, then the copy goes until end of file. All the data read from `inchan` is copied to `outchan`. Without the `-command` option, `fcopy` blocks until the copy is complete and returns the number of bytes written to `outchan`.

The `-command` argument makes `fcopy` work in the background. In this case it returns immediately and the `callback` is invoked later when the copy completes. The `callback` is called with one or two additional arguments that indicates how many bytes were written to `outchan`. If an error occurred during the background copy, the second argument is the error string associated with the error. With a background copy, it is not necessary to put `inchan` or `outchan` into non-blocking mode; the `fcopy` command takes care of that automatically. However, it is necessary to enter the event loop by using the `vwait` command or by using Tk.
You are not allowed to do other I/O operations with \textit{inchan} or \textit{outchan} during a background \texttt{fcopy}. If either \textit{inchan} or \textit{outchan} get closed while the copy is in progress, the current copy is stopped and the command callback is \textit{not} made. If \textit{inchan} is closed, then all data already queued for \textit{outchan} is written out.

Note that \textit{inchan} can become readable during a background copy. You should turn off any \texttt{fileevent} handlers during a background copy so those handlers do not interfere with the copy. Any I/O attempted by a \texttt{fileevent} handler will get a “channel busy” error.

\texttt{Fc}opy translates end-of-line sequences in \textit{inchan} and \textit{outchan} according to the \texttt{-translation} option for these channels. See the manual entry for \texttt{fconfigure} for details on the \texttt{-translation} option. The translations mean that the number of bytes read from \textit{inchan} can be different than the number of bytes written to \textit{outchan}. Only the number of bytes written to \textit{outchan} is reported, either as the return value of a synchronous \texttt{fcopy} or as the argument to the callback for an asynchronous \texttt{fcopy}.

\texttt{Fc}opy obeys the encodings and character translations configured for the channels. This means that the incoming characters are converted internally first UTF-8 and then into the encoding of the channel \texttt{fcopy} writes to. See the manual entry for \texttt{fconfigure} for details on the \texttt{-encoding} and \texttt{-translation} options. No conversion is done if both channels are set to encoding “binary” and have matching translations. If only the output channel is set to encoding “binary” the system will write the internal UTF-8 representation of the incoming characters. If only the input channel is set to encoding “binary” the system will assume that the incoming bytes are valid UTF-8 characters and convert them according to the output encoding. The behaviour of the system for bytes which are not valid UTF-8 characters is undefined in this case.

\textbf{EXAMPLES}

The first example transfers the contents of one channel exactly to another. Note that when copying one file to another, it is better to use \texttt{file copy} which also copies file metadata (e.g. the file access
permissions) where possible.

```bash
fconfigure $in -translation binary
fconfigure $out -translation binary
fcopy $in $out
```

This second example shows how the callback gets passed the number of bytes transferred. It also uses vwait to put the application into the event loop. Of course, this simplified example could be done without the command callback.

```bash
proc Cleanup {in out bytes {error {}}} {
    global total
    set total $bytes
    close $in
    close $out
    if {([string length $error] != 0) { # error occurred during the copy
        }
    }
}
set in [open $file1]
set out [socket $server $port]
fcopy $in $out -command [list Cleanup $in $out]
vwait total
```

The third example copies in chunks and tests for end of file in the command callback.

```bash
proc CopyMore {in out chunk bytes {error {}}} {
    global total done
    incr total $bytes
    if {(([string length $error] != 0) || [eof $in])
        set done $total
```
close $in
close $out
} else {
    fcopy $in $out -size $chunk \n    -command [list CopyMore $in $out $chunk]
}
set in [open $file1]
set out [socket $server $port]
set chunk 1024
set total 0
fcopy $in $out -size $chunk \n    -command [list CopyMore $in $out $chunk]
vwait done

SEE ALSO
eof, fblocked, fconfigure, file

KEYWORDS
blocking, channel, end of line, end of file, nonblocking, read, translation

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME
Irang - Return one or more adjacent elements from a list

SYNOPSIS
Irang list first last

DESCRIPTION
List must be a valid Tcl list. This command will return a new list consisting of elements first through last, inclusive. The index values first and last are interpreted the same as index values for the command string index, supporting simple index arithmetic and indices relative to the end of the list. If first is less than zero, it is treated as if it were zero. If last is greater than or equal to the number of elements in the list, then it is treated as if it were end. If first is greater than last then an empty string is returned. Note: “Irang list first first” does not always produce the same result as “lindex list first” (although it often does for simple fields that are not enclosed in braces); it does, however, produce exactly the same results as “list [lindex list first]"

EXAMPLES
Selecting the first two elements:

% Irang {a b c d e} 0 1
a b

Selecting the last three elements:
% \texttt{lrange \{a b c d e\} end-2 end}
\begin{verbatim}
c d e
\end{verbatim}

Selecting everything except the first and last element:

% \texttt{lrange \{a b c d e\} 1 end-1}
\begin{verbatim}
b c d
\end{verbatim}

Selecting a single element with \texttt{lrange} is not the same as doing so with \texttt{lindex}:

% \texttt{set var \{some \{elements to\} select\}
\begin{verbatim}
some \{elements to\} select
\end{verbatim}
% \texttt{lindex $var 1}
\begin{verbatim}
elements to
\end{verbatim}
% \texttt{lrange $var 1 1}
\begin{verbatim}
\{elements to\}
\end{verbatim}

\textbf{SEE ALSO}

\texttt{list, lappend, lindex, linsert, llength, lsearch, lset, lreplace, lsort, string}

\textbf{KEYWORDS}

element, list, range, sublist

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 2001 Kevin B. Kenny <kennykb(at)acm.org>. All rights reserved.
NAME
refchan - Command handler API of reflected channels, version 1

SYNOPSIS

DESCRIPTION

MANDATORY SUBCOMMANDS

  cmdPrefix initialize channelId mode
  cmdPrefix finalize channelId
  cmdPrefix watch channelId eventspec

OPTIONAL SUBCOMMANDS

  cmdPrefix read channelId count
  cmdPrefix write channelId data
  cmdPrefix seek channelId offset base
    start
    current
    end
  cmdPrefix configure channelId option value
  cmdPrefix cget channelId option
  cmdPrefix cgetall channelId
  cmdPrefix blocking channelId mode

NOTES

SEE ALSO

KEYWORDS
DESCRIPTION

The Tcl-level handler for a reflected channel has to be a command with subcommands (termed an ensemble, as it is a command such as that created by namespace ensemble create, though the implementation of handlers for reflected channel is not tied to namespace ensembles in any way). Note that cmdPrefix is whatever was specified in the call to chan create, and may consist of multiple arguments; this will be expanded to multiple words in place of the prefix.

Of all the possible subcommands, the handler must support initialize, finalize, and watch. Support for the other subcommands is optional.

MANDATORY SUBCOMMANDS

cmdPrefix initialize channelId mode
  An invocation of this subcommand will be the first call the cmdPrefix will receive for the specified new channelId. It is the responsibility of this subcommand to set up any internal data structures required to keep track of the channel and its state.

  The return value of the method has to be a list containing the names of all subcommands supported by the cmdPrefix. This also tells the Tcl core which version of the API for reflected channels is used by this command handler.

  Any error thrown by the method will abort the creation of the channel and no channel will be created. The thrown error will appear as error thrown by chan create. Any exception other than an error (e.g. break, etc.) is treated as (and converted to) an error.

  Note: If the creation of the channel was aborted due to failures here, then the finalize subcommand will not be called.

  The mode argument tells the handler whether the channel was opened for reading, writing, or both. It is a list containing any of the strings read or write. The list will always contain at least one element.
The subcommand must throw an error if the chosen mode is not supported by the `cmdPrefix`.

**cmdPrefix `finalize channelId`**
An invocation of this subcommand will be the last call the `cmdPrefix` will receive for the specified `channelId`. It will be generated just before the destruction of the data structures of the channel held by the Tcl core. The command handler *must not* access the `channelId` anymore in no way. Upon this subcommand being called, any internal resources allocated to this channel must be cleaned up.

The return value of this subcommand is ignored.

If the subcommand throws an error the command which caused its invocation (usually `close`) will appear to have thrown this error. Any exception beyond `error` (e.g. `break`, etc.) is treated as (and converted to) an error.

This subcommand is not invoked if the creation of the channel was aborted during `initialize` (See above).

**cmdPrefix `watch channelId eventspec`**
This subcommand notifies the `cmdPrefix` that the specified `channelId` is interested in the events listed in the `eventspec`. This argument is a list containing any of `read` and `write`. The list may be empty, which signals that the channel does not wish to be notified of any events. In that situation, the handler should disable event generation completely.

**Warning:** Any return value of the subcommand is ignored. This includes all errors thrown by the subcommand, break, continue, and custom return codes.

This subcommand interacts with `chan postevent`. Trying to post an event which was not listed in the last call to `watch` will cause `chan postevent` to throw an error.
OPTIMAL SUBCOMMANDS

`cmdPrefix read channelld count`
This *optional* subcommand is called when the user requests data from the channel `channelld`. `count` specifies how many *bytes* have been requested. If the subcommand is not supported then it is not possible to read from the channel handled by the command.

The return value of this subcommand is taken as the requested data *bytes*. If the returned data contains more bytes than requested, an error will be signaled and later thrown by the command which performed the read (usually `gets` or `read`). However, returning fewer bytes than requested is acceptable.

Note that returning nothing (0 bytes) is a signal to the higher layers that *EOF* has been reached on the channel. To signal that the channel is out of data right now, but has not yet reached *EOF*, it is necessary to throw the error "EAGAIN", i.e. to either

```
return -code error EAGAIN
```

or

```
error EAGAIN
```

For extensibility any error whose value is a negative integer number will cause the higher layers to set the C-level variable "errno" to the absolute value of this number, signaling a system error. This means that both

```
return -code error -11
```

and
are equivalent to the examples above, using the more readable string "EAGAIN". No other error value has such a mapping to a symbolic string.

If the subcommand throws any other error, the command which caused its invocation (usually puts, or read) will appear to have thrown this error. Any exception beyond error, (e.g. break, etc.) is treated as and converted to an error.

**cmdPrefix write channelId data**

This optional subcommand is called when the user writes data to the channel `channelId`. The `data` argument contains bytes, not characters. Any type of transformation (EOL, encoding) configured for the channel has already been applied at this point. If this subcommand is not supported then it is not possible to write to the channel handled by the command.

The return value of the subcommand is taken as the number of bytes written by the channel. Anything non-numeric will cause an error to be signaled and later thrown by the command which performed the write. A negative value implies that the write failed. Returning a value greater than the number of bytes given to the handler, or zero, is forbidden and will cause the Tcl core to throw an error.

If the subcommand throws an error the command which caused its invocation (usually puts) will appear to have thrown this error. Any exception beyond error (e.g. break, etc.) is treated as and converted to an error.

**cmdPrefix seek channelId offset base**

This optional subcommand is responsible for the handling of seek and tell requests on the channel `channelId`. If it is not supported then seeking will not be possible for the channel.
The *base* argument is one of

**start**

Seeking is relative to the beginning of the channel.

**current**

Seeking is relative to the current seek position.

**end**

Seeking is relative to the end of the channel.

The *base* argument of the builtin *chan seek* command takes the same names.

The *offset* is an integer number specifying the amount of *bytes* to seek forward or backward. A positive number should seek forward, and a negative number should seek backward.

A channel may provide only limited seeking. For example sockets can seek forward, but not backward.

The return value of the subcommand is taken as the (new) location of the channel, counted from the start. This has to be an integer number greater than or equal to zero.

If the subcommand throws an error the command which caused its invocation (usually *seek*, or *tell*) will appear to have thrown this error. Any exception beyond *error* (e.g. *break*, etc.) is treated as and converted to an error.

The offset/base combination of 0/current signals a *tell* request, i.e. seek nothing relative to the current location, making the new location identical to the current one, which is then returned.

**cmdPrefix configure channelld option value**

This *optional* subcommand is for setting the type-specific options of channel *channelld*. The *option* argument indicates the option to be written, and the *value* argument indicates the value to set the option to.
This subcommand will never try to update more than one option at a time; that is behavior implemented in the Tcl channel core.

The return value of the subcommand is ignored.

If the subcommand throws an error the command which performed the (re)configuration or query (usually fconfigure or chan configure) will appear to have thrown this error. Any exception beyond error (e.g. break, etc.) is treated as and converted to an error.

```
cmdPrefix cget channelId option
```
This optional subcommand is used when reading a single type-specific option of channel channelId. If this subcommand is supported then the subcommand cgetall must be supported as well.

The subcommand should return the value of the specified option.

If the subcommand throws an error, the command which performed the (re)configuration or query (usually fconfigure) will appear to have thrown this error. Any exception beyond error (e.g. break, etc.) is treated as and converted to an error.

```
cmdPrefix cgetall channelId
```
This optional subcommand is used for reading all type-specific options of channel channelId. If this subcommand is supported then the subcommand cget has to be supported as well.

The subcommand should return a list of all options and their values. This list must have an even number of elements.

If the subcommand throws an error the command which performed the (re)configuration or query (usually fconfigure) will appear to have thrown this error. Any exception beyond error (e.g. break, etc.) is treated as and converted to an error.

```
cmdPrefix blocking channelId mode
```
This optional subcommand handles changes to the blocking mode
of the channel \textit{channelId}. The \textit{mode} is a boolean flag. A true value means that the channel has to be set to blocking, and a false value means that the channel should be non-blocking.

The return value of the subcommand is ignored.

If the subcommand throws an error the command which caused its invocation (usually \texttt{fconfigure}) will appear to have thrown this error. Any exception beyond \textit{error} (e.g. \textit{break}, etc.) is treated as and converted to an error.

\textbf{NOTES}

Some of the functions supported in channels defined in Tcl's C interface are not available to channels reflected to the Tcl level.

The function \texttt{Tcl\_DriverGetHandleProc} is not supported; i.e. reflected channels do not have OS specific handles.

The function \texttt{Tcl\_DriverHandlerProc} is not supported. This driver function is relevant only for stacked channels, i.e. transformations. Reflected channels are always base channels, not transformations.

The function \texttt{Tcl\_DriverFlushProc} is not supported. This is because the current generic I/O layer of Tcl does not use this function anywhere at all. Therefore support at the Tcl level makes no sense either. This may be altered in the future (through extending the API defined here and changing its version number) should the function be used at some time in the future.

\textbf{SEE ALSO}

\texttt{chan}

\textbf{KEYWORDS}

\texttt{channel}, \texttt{reflection}

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

tell - Return current access position for an open channel

SYNOPSIS

tell channelId

DESCRIPTION

Returns an integer string giving the current access position in channelId. This value returned is a byte offset that can be passed to seek in order to set the channel to a particular position. Note that this value is in terms of bytes, not characters like read. The value returned is -1 for channels that do not support seeking.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

EXAMPLE

Read a line from a file channel only if it starts with foobar:

```tcl
# Save the offset in case we need to undo the read..
set offset [tell $chan]
if {([read $chan 6] eq "foobar") {
    gets $chan line
} else {
    set line {}
```
# Undo the read...
seek $chan $offset
}

SEE ALSO

file, open, close, gets, seek, Tcl_StandardChannels

KEYWORDS

access position, channel, seeking

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
  file - Manipulate file names and attributes

SYNOPSIS

DESCRIPTION

  file atime name ?time?
  file attributes name
  file attributes name ?option?
  file attributes name ?option value option value...?
  file channels ?pattern?
  file copy ?-force? ?--? source target
  file copy ?-force? ?--? source ?source ...? targetDir
  file delete ?-force? ?--? pathname ?pathname ... ?
  file dirname name
  file executable name
  file exists name
  file extension name
  file isdirectory name
  file isfile name
  file join name ?name ...?
  file link ?-linktype? linkName ?target?
  file lstat name varName
  file mkdir dir ?dir ...?
  file mtime name ?time?
  file nativename name
  file normalize name
  file owned name
  file pathtype name
  file readable name
  file readlink name
  file rename ?-force? ?--? source target
  file rename ?-force? ?--? source ?source ...? targetDir
PORTABILITY ISSUES
Unix
EXAMPLES
SEE ALSO
KEYWORDS

NAME

file - Manipulate file names and attributes

SYNOPSIS

file option name ?arg arg ...?

DESCRIPTION

This command provides several operations on a file's name or attributes. Name is the name of a file; if it starts with a tilde, then tilde substitution is done before executing the command (see the manual entry for filename for details). Option indicates what to do with the file name. Any unique abbreviation for option is acceptable. The valid options are:

file atime name ?time?
Returns a decimal string giving the time at which file name was last accessed. If time is specified, it is an access time to set for the file. The time is measured in the standard POSIX fashion as seconds from a fixed starting time (often January 1, 1970). If the file does
not exist or its access time cannot be queried or set then an error is generated. On Windows, FAT file systems do not support access time.

file attributes name

file attributes name ?option?

file attributes name ?option value option value...?

This subcommand returns or sets platform specific values associated with a file. The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

On Unix, -group gets or sets the group name for the file. A group id can be given to the command, but it returns a group name. -owner gets or sets the user name of the owner of the file. The command returns the owner name, but the numerical id can be passed when setting the owner. -permissions sets or retrieves the octal code that chmod(1) uses. This command does also has limited support for setting using the symbolic attributes for chmod(1), of the form [ugo]?[+-=][rwxst],[...], where multiple symbolic attributes can be separated by commas (example: u+s,go-rw add sticky bit for user, remove read and write permissions for group and other). A simplified ls style string, of the form rwxrwxrwx (must be 9 characters), is also supported (example: rwxr-xr-t is equivalent to 01755). On versions of Unix supporting file flags, -readonly gives the value or sets or clears the readonly attribute of the file, i.e. the user immutable flag uchg to chflags(1).

On Windows, -archive gives the value or sets or clears the archive attribute of the file. -hidden gives the value or sets or clears the hidden attribute of the file. -longname will expand each path element to its long version. This attribute cannot be set. -readonly gives the value or sets or clears the readonly attribute of the file. -shortname gives a string where every path element is replaced with its short (8.3) version of the name. This attribute cannot be set.
-**system** gives or sets or clears the value of the system attribute of the file.

On Mac OS X and Darwin, **-creator** gives or sets the Finder creator type of the file. **-hidden** gives or sets or clears the hidden attribute of the file. **-readonly** gives or sets or clears the readonly attribute of the file. **-rsrclength** gives the length of the resource fork of the file, this attribute can only be set to the value 0, which results in the resource fork being stripped off the file.

**file channels ??pattern?**
If *pattern* is not specified, returns a list of names of all registered open channels in this interpreter. If *pattern* is specified, only those names matching *pattern* are returned. Matching is determined using the same rules as for **string match**.

**file copy ??-force? ?--? source target**

**file copy ??-force? ?--? source ??source ??targetDir**
The first form makes a copy of the file or directory *source* under the pathname *target*. If *target* is an existing directory, then the second form is used. The second form makes a copy inside *targetDir* of each *source* file listed. If a directory is specified as a *source*, then the contents of the directory will be recursively copied into *targetDir*. Existing files will not be overwritten unless the **-force** option is specified (when Tcl will also attempt to adjust permissions on the destination file or directory if that is necessary to allow the copy to proceed). When copying within a single filesystem, **file copy** will copy soft links (i.e. the links themselves are copied, not the things they point to). Trying to overwrite a non-empty directory, overwrite a directory with a file, or overwrite a file with a directory will all result in errors even if **-force** was specified. Arguments are processed in the order specified, halting at the first error, if any. A **--** marks the end of switches; the argument following the **--** will be treated as a *source* even if it starts with a **-**.

**file delete ??-force? ?--? pathname ??pathname ??**
Removes the file or directory specified by each *pathname*
argument. Non-empty directories will be removed only if the `-force` option is specified. When operating on symbolic links, the links themselves will be deleted, not the objects they point to. Trying to delete a non-existent file is not considered an error. Trying to delete a read-only file will cause the file to be deleted, even if the `-force` flags is not specified. If the `-force` option is specified on a directory, Tcl will attempt both to change permissions and move the current directory “pwd” out of the given path if that is necessary to allow the deletion to proceed. Arguments are processed in the order specified, halting at the first error, if any. A `--` marks the end of switches; the argument following the `--` will be treated as a `pathname` even if it starts with a `-.

**file dirname name**

Returns a name comprised of all of the path components in name excluding the last element. If `name` is a relative file name and only contains one path element, then returns “.”. If `name` refers to a root directory, then the root directory is returned. For example,

```plaintext
file dirname c:/
```

returns `c:/`.  

Note that tilde substitution will only be performed if it is necessary to complete the command. For example,

```plaintext
file dirname ~/src/foo.c
```

returns `~/src`, whereas

```plaintext
file dirname ~
```

returns `/home` (or something similar).
file executable name
   Returns 1 if file name is executable by the current user, 0 otherwise.

file exists name
   Returns 1 if file name exists and the current user has search privileges for the directories leading to it, 0 otherwise.

file extension name
   Returns all of the characters in name after and including the last dot in the last element of name. If there is no dot in the last element of name then returns the empty string.

file isdirectory name
   Returns 1 if file name is a directory, 0 otherwise.

file isfile name
   Returns 1 if file name is a regular file, 0 otherwise.

file join name ?name ...
   Takes one or more file names and combines them, using the correct path separator for the current platform. If a particular name is relative, then it will be joined to the previous file name argument. Otherwise, any earlier arguments will be discarded, and joining will proceed from the current argument. For example,

   \texttt{file join a b /foo bar}

   returns /foo/bar.

   Note that any of the names can contain separators, and that the result is always canonical for the current platform: / for Unix and Windows.

file link ?-linktype? linkName ?target?
   If only one argument is given, that argument is assumed to be linkName, and this command returns the value of the link given by
*linkName* (i.e. the name of the file it points to). If *linkName* is not a link or its value cannot be read (as, for example, seems to be the case with hard links, which look just like ordinary files), then an error is returned.

If 2 arguments are given, then these are assumed to be *linkName* and *target*. If *linkName* already exists, or if *target* does not exist, an error will be returned. Otherwise, Tcl creates a new link called *linkName* which points to the existing filesystem object at *target* (which is also the returned value), where the type of the link is platform-specific (on Unix a symbolic link will be the default). This is useful for the case where the user wishes to create a link in a cross-platform way, and does not care what type of link is created.

If the user wishes to make a link of a specific type only, (and signal an error if for some reason that is not possible), then the optional `-linktype` argument should be given. Accepted values for `-linktype` are `"-symbolic"` and `"-hard"`.

On Unix, symbolic links can be made to relative paths, and those paths must be relative to the actual *linkName*'s location (not to the cwd), but on all other platforms where relative links are not supported, target paths will always be converted to absolute, normalized form before the link is created (and therefore relative paths are interpreted as relative to the cwd). Furthermore, `"~user"` paths are always expanded to absolute form. When creating links on filesystems that either do not support any links, or do not support the specific type requested, an error message will be returned. In particular Windows 95, 98 and ME do not support any links at present, but most Unix platforms support both symbolic and hard links (the latter for files only) and Windows NT/2000/XP (on NTFS drives) support symbolic directory links and hard file links.

**file lstat name varName**

Same as **stat** option (see below) except uses the *lstat* kernel call instead of *stat*. This means that if *name* refers to a symbolic link the information returned in *varName* is for the link rather than the file it refers to. On systems that do not support symbolic links this option
behaves exactly the same as the stat option.

**file mkdir dir ?dir ...?**

Creates each directory specified. For each pathname dir specified, this command will create all non-existing parent directories as well as dir itself. If an existing directory is specified, then no action is taken and no error is returned. Trying to overwrite an existing file with a directory will result in an error. Arguments are processed in the order specified, halting at the first error, if any.

**file mtime name ?time?**

Returns a decimal string giving the time at which file name was last modified. If time is specified, it is a modification time to set for the file (equivalent to Unix touch). The time is measured in the standard POSIX fashion as seconds from a fixed starting time (often January 1, 1970). If the file does not exist or its modified time cannot be queried or set then an error is generated.

**file nativename name**

Returns the platform-specific name of the file. This is useful if the filename is needed to pass to a platform-specific call, such as to a subprocess via exec under Windows (see EXAMPLES below).

**file normalize name**

Returns a unique normalized path representation for the file-system object (file, directory, link, etc), whose string value can be used as a unique identifier for it. A normalized path is an absolute path which has all “../” and “./” removed. Also it is one which is in the “standard” format for the native platform. On Unix, this means the segments leading up to the path must be free of symbolic links/aliases (but the very last path component may be a symbolic link), and on Windows it also means we want the long form with that form's case-dependence (which gives us a unique, case-dependent path). The one exception concerning the last link in the path is necessary, because Tcl or the user may wish to operate on the actual symbolic link itself (for example file delete, file rename, file copy are defined to operate on symbolic links, not on the things that they point to).
file owned name
Returns 1 if file name is owned by the current user, 0 otherwise.

file pathtype name
Returns one of absolute, relative, volumerelative. If name refers to a specific file on a specific volume, the path type will be absolute. If name refers to a file relative to the current working directory, then the path type will be relative. If name refers to a file relative to the current working directory on a specified volume, or to a specific file on the current working volume, then the path type is volumerelative.

file readable name
Returns 1 if file name is readable by the current user, 0 otherwise.

file readlink name
Returns the value of the symbolic link given by name (i.e. the name of the file it points to). If name is not a symbolic link or its value cannot be read, then an error is returned. On systems that do not support symbolic links this option is undefined.

file rename ?-force? ?--? source target

file rename ?-force? ?--? source ?source ...? targetDir
The first form takes the file or directory specified by pathnamensource and renames it to target, moving the file if the pathnametarget specifies a name in a different directory. If target is an existing directory, then the second form is used. The second form moves each source file or directory into the directory targetDir. Existing files will not be overwritten unless the -force option is specified. When operating inside a single filesystem, Tcl will rename symbolic links rather than the things that they point to. Trying to overwrite a non-empty directory, overwrite a directory with a file, or a file with a directory will all result in errors. Arguments are processed in the order specified, halting at the first error, if any. A -- marks the end of switches; the argument following the -- will be treated as a source even if it starts with a -.
file rootname name
Returns all of the characters in name up to but not including the last “.” character in the last component of name. If the last component of name does not contain a dot, then returns name.

file separator ?name?
If no argument is given, returns the character which is used to separate path segments for native files on this platform. If a path is given, the filesystem responsible for that path is asked to return its separator character. If no file system accepts name, an error is generated.

file size name
Returns a decimal string giving the size of file name in bytes. If the file does not exist or its size cannot be queried then an error is generated.

file split name
Returns a list whose elements are the path components in name. The first element of the list will have the same path type as name. All other elements will be relative. Path separators will be discarded unless they are needed ensure that an element is unambiguously relative. For example, under Unix

```
file split /foo/~bar/baz
```

returns /foo .~bar baz to ensure that later commands that use the third component do not attempt to perform tilde substitution.

file stat name varName
Invokes the stat kernel call on name, and uses the variable given by varName to hold information returned from the kernel call. VarName is treated as an array variable, and the following elements of that variable are set: atime, ctime, dev, gid, ino, mode, mtime, nlink, size, type, uid. Each element except type is a decimal string with the value of the corresponding field from the
stat return structure; see the manual entry for stat for details on the meanings of the values. The type element gives the type of the file in the same form returned by the command file type. This command returns an empty string.

file system name
Returns a list of one or two elements, the first of which is the name of the filesystem to use for the file, and the second, if given, an arbitrary string representing the filesystem-specific nature or type of the location within that filesystem. If a filesystem only supports one type of file, the second element may not be supplied. For example the native files have a first element “native”, and a second element which when given is a platform-specific type name for the file's system (e.g. “NTFS”, “FAT”, on Windows). A generic virtual file system might return the list “vfs ftp” to represent a file on a remote ftp site mounted as a virtual filesystem through an extension called “vfs”. If the file does not belong to any filesystem, an error is generated.

file tail name
Returns all of the characters in the last filesystem component of name. Any trailing directory separator in name is ignored. If name contains no separators then returns name. So, file tail a/b, file tail a/b/ and file tail b all return b.

file type name
Returns a string giving the type of file name, which will be one of file, directory, characterSpecial, blockSpecial, fifo, link, or socket.

file volumes
Returns the absolute paths to the volumes mounted on the system, as a proper Tcl list. Without any virtual filesystems mounted as root volumes, on UNIX, the command will always return “/”, since all filesystems are locally mounted. On Windows, it will return a list of the available local drives (e.g. “a:/ c:/”). If any virtual filesystem has mounted additional volumes, they will be in the returned list.
file writable name

   Returns 1 if file name is writable by the current user, 0 otherwise.

**PORTABILITY ISSUES**

Unix

   These commands always operate using the real user and group
   identifiers, not the effective ones.

**EXAMPLES**

This procedure shows how to search for C files in a given directory that
have a correspondingly-named object file in the current directory:

```tcl
proc findMatchingCFiles {dir} {
   set files {}
   switch $::tcl_platform(platform) {
      windows {
         set ext .obj
      }
      unix {
         set ext .o
      }
   }
   foreach file [glob -nocomplain -directory $dir *.c]
      set objectFile [file tail [file rootname $file]
      if {[file exists $objectFile]} {
         lappend files $file
      }
   }
   return $files
}
```

Rename a file and leave a symbolic link pointing from the old location to
the new place:
set oldName foobar.txt
set newName foo/bar.txt
#
# Make sure that where we're going to move to exists
if {![file isdirectory [file dirname $newName]]} {
  file mkdir [file dirname $newName]
}
file rename $oldName $newName
file link -symbolic $oldName $newName

On Windows, a file can be “started” easily enough (equivalent to
double-clicking on it in the Explorer interface) but the name passed to
the operating system must be in native format:

exec {*}[auto_execok start] {} [file nativename ~/ex

SEE ALSO

filename, open, close, eof, gets, tell, seek, fblocked, flush

KEYWORDS

attributes, copy files, delete files, directory, file, move files, name,
rename files, stat

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

lrepeat - Build a list by repeating elements

SYNOPSIS

lrepeat number element1 ?element2 element3 ...?

DESCRIPTION

The **lrepeat** command creates a list of size \( number \times number \) of elements by repeating \( number \) times the sequence of elements \( element1 \ \text{element2} \ \text{...} \). \( number \) must be a positive integer, \( elementn \) can be any Tcl value. Note that **lrepeat 1 arg ...** is identical to **list arg ...**, though the **arg** is required with **lrepeat**.

EXAMPLES

```
lrepeat 3 a
    → a a a
lrepeat 3 [lrepeat 3 0]
    → {0 0 0} {0 0 0} {0 0 0}
lrepeat 3 a b c
    → a b c a b c a b c
lrepeat 3 [lrepeat 2 a] b c
    → {a a} b c {a a} b c {a a} b c
```

SEE ALSO
list, lappend, linsert, llength, lset

KEYWORDS

element, index, list

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2003 by Simon Geard. All rights reserved.
NAME
regexp - Match a regular expression against a string

SYNOPSIS

DESCRIPTION
- about
- expanded
- indices
- line
- linestop
- lineanchor
- nocase
- all
- inline
- start index

EXAMPLES
SEE ALSO
KEYWORDS

NAME
regexp - Match a regular expression against a string

SYNOPSIS

regexp \(?switches?\) exp string \(?matchVar?\) \(?subMatchVar\) subMatchVar ...

DESCRIPTION

Determines whether the regular expression exp matches part or all of string and returns 1 if it does, 0 if it does not, unless -inline is specified
(see below). (Regular expression matching is described in the
re_syntax reference page.)

If additional arguments are specified after string then they are treated
as the names of variables in which to return information about which
part(s) of string matched exp. MatchVar will be set to the range of string
that matched all of exp. The first subMatchVar will contain the
characters in string that matched the leftmost parenthesized
subexpression within exp, the next subMatchVar will contain the
characters that matched the next parenthesized subexpression to the
right in exp, and so on.

If the initial arguments to regexp start with - then they are treated as
switches. The following switches are currently supported:

- **about**
  Instead of attempting to match the regular expression, returns a list
  containing information about the regular expression. The first
  element of the list is a subexpression count. The second element is
  a list of property names that describe various attributes of the
  regular expression. This switch is primarily intended for debugging
  purposes.

- **expanded**
  Enables use of the expanded regular expression syntax where
  whitespace and comments are ignored. This is the same as
  specifying the (?)x embedded option (see the re_syntax manual
  page).

- **indices**
  Changes what is stored in the subMatchVars. Instead of storing the
  matching characters from string, each variable will contain a list of
two decimal strings giving the indices in string of the first and last
  characters in the matching range of characters.

- **line**
  Enables newline-sensitive matching. By default, newline is a
  completely ordinary character with no special meaning. With this
flag, “[^” bracket expressions and “.” never match newline, “^” matches an empty string after any newline in addition to its normal function, and “$” matches an empty string before any newline in addition to its normal function. This flag is equivalent to specifying both -linestop and -lineanchor, or the (?n) embedded option (see the re_syntax manual page).

-linestop
Changes the behavior of “[^” bracket expressions and “.” so that they stop at newlines. This is the same as specifying the (?p) embedded option (see the re_syntax manual page).

-lineanchor
Changes the behavior of “^” and “$” (the “anchors”) so they match the beginning and end of a line respectively. This is the same as specifying the (?w) embedded option (see the re_syntax manual page).

-nocase
Causes upper-case characters in string to be treated as lower case during the matching process.

-all
Causes the regular expression to be matched as many times as possible in the string, returning the total number of matches found. If this is specified with match variables, they will contain information for the last match only.

-inline
Causes the command to return, as a list, the data that would otherwise be placed in match variables. When using -inline, match variables may not be specified. If used with -all, the list will be concatenated at each iteration, such that a flat list is always returned. For each match iteration, the command will append the overall match data, plus one element for each subexpression in the regular expression. Examples are:
**-start index**

Specifies a character index offset into the string to start matching the regular expression at. The `index` value is interpreted in the same manner as the `index` argument to `string index`. When using this switch, “^” will not match the beginning of the line, and \A will still match the start of the string at `index`. If `-indices` is specified, the indices will be indexed starting from the absolute beginning of the input string. `index` will be constrained to the bounds of the input string.

```
--
```

Marks the end of switches. The argument following this one will be treated as `exp` even if it starts with a `-`.

If there are more `subMatchVars` than parenthesized subexpressions within `exp`, or if a particular subexpression in `exp` does not match the string (e.g. because it was in a portion of the expression that was not matched), then the corresponding `subMatchVar` will be set to “-1 -1” if `-indices` has been specified or to an empty string otherwise.

---

**EXAMPLES**

Find the first occurrence of a word starting with `foo` in a string that is not actually an instance of `foobar`, and get the letters following it up to the end of the word into a variable:

```
regexp {\mfoo(?!bar\M)(\w*)} $string -> restOfWord
```

Note that the whole matched substring has been placed in the variable
“->”, which is a name chosen to look nice given that we are not actually interested in its contents.

Find the index of the word badger (in any case) within a string and store that in the variable location:

```
regexp -indices {(?i)\mbadger\M} $string location
```

This could also be written as a basic regular expression (as opposed to using the default syntax of advanced regular expressions) match by prefixing the expression with a suitable flag:

```
regexp -indices {(?ib)\<badger\>} $string location
```

This counts the number of octal digits in a string:

```
regexp -all {[0-7]} $string
```

This lists all words (consisting of all sequences of non-whitespace characters) in a string, and is useful as a more powerful version of the `split` command:

```
regexp -all -inline {\S+} $string
```

**SEE ALSO**

`re_syntax`, `regsub`, `string`

**KEYWORDS**

`match`, `parsing`, `pattern`, `regular_expression`, `splitting`, `string`
time - Time the execution of a script

SYNOPSIS

time script ?count?

DESCRIPTION

This command will call the Tcl interpreter count times to evaluate script (or once if count is not specified). It will then return a string of the form

503 microseconds per iteration

which indicates the average amount of time required per iteration, in microseconds. Time is measured in elapsed time, not CPU time.

EXAMPLE

Estimate how long it takes for a simple Tcl for loop to count to a thousand:

```
time {
    for {set i 0} {$i<1000} {incr i} {
        # empty body
    }
}
```
NAME

fileevent - Execute a script when a channel becomes readable or writable

SYNOPSIS

fileevent channelId readable ?script?  
fileevent channelId writable ?script?

DESCRIPTION

This command is used to create file event handlers. A file event handler is a binding between a channel and a script, such that the script is evaluated whenever the channel becomes readable or writable. File event handlers are most commonly used to allow data to be received from another process on an event-driven basis, so that the receiver can continue to interact with the user while waiting for the data to arrive. If an application invokes gets or read on a blocking channel when there is no input data available, the process will block; until the input data arrives, it will not be able to service other events, so it will appear to the user to “freeze up”. With fileevent, the process can tell when data is present and only invoke gets or read when they will not block.

The channelId argument to fileevent refers to an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

If the script argument is specified, then fileevent creates a new event handler: script will be evaluated whenever the channel becomes readable or writable (depending on the second argument to fileevent).
In this case fileevent returns an empty string. The readable and writable event handlers for a file are independent, and may be created and deleted separately. However, there may be at most one readable and one writable handler for a file at a given time in a given interpreter. If fileevent is called when the specified handler already exists in the invoking interpreter, the new script replaces the old one.

If the script argument is not specified, fileevent returns the current script for channelId, or an empty string if there is none. If the script argument is specified as an empty string then the event handler is deleted, so that no script will be invoked. A file event handler is also deleted automatically whenever its channel is closed or its interpreter is deleted.

A channel is considered to be readable if there is unread data available on the underlying device. A channel is also considered to be readable if there is unread data in an input buffer, except in the special case where the most recent attempt to read from the channel was a gets call that could not find a complete line in the input buffer. This feature allows a file to be read a line at a time in nonblocking mode using events. A channel is also considered to be readable if an end of file or error condition is present on the underlying file or device. It is important for script to check for these conditions and handle them appropriately; for example, if there is no special check for end of file, an infinite loop may occur where script reads no data, returns, and is immediately invoked again.

A channel is considered to be writable if at least one byte of data can be written to the underlying file or device without blocking, or if an error condition is present on the underlying file or device.

Event-driven I/O works best for channels that have been placed into nonblocking mode with the fconfigure command. In blocking mode, a puts command may block if you give it more data than the underlying file or device can accept, and a gets or read command will block if you attempt to read more data than is ready; no events will be processed while the commands block. In nonblocking mode puts, read, and gets never block. See the documentation for the individual commands for
information on how they handle blocking and nonblocking channels.

The script for a file event is executed at global level (outside the context of any Tcl procedure) in the interpreter in which the fileevent command was invoked. If an error occurs while executing the script then the command registered with interp bgerror is used to report the error. In addition, the file event handler is deleted if it ever returns an error; this is done in order to prevent infinite loops due to buggy handlers.

**EXAMPLE**

In this setup GetData will be called with the channel as an argument whenever $chan becomes readable.

```tcl
proc GetData {chan} {
    if { ![eof $chan] } {
        puts [gets $chan]
    }
}

fileevent $chan readable [list GetData $chan]
```

**CREDITS**

fileevent is based on the addinput command created by Mark Diekhans.

**SEE ALSO**

fconfigure, gets, interp, puts, read, Tcl_StandardChannels

**KEYWORDS**

asynchronous I/O, blocking, channel, event handler, nonblocking, readable, script, writable.
NAME
Ireplace - Replace elements in a list with new elements

SYNOPSIS
Ireplace list first last ?element element ...?

DESCRIPTION
Ireplace returns a new list formed by replacing one or more elements of list with the element arguments. first and last are index values specifying the first and last elements of the range to replace. The index values first and last are interpreted the same as index values for the command string index, supporting simple index arithmetic and indices relative to the end of the list. 0 refers to the first element of the list, and end refers to the last element of the list. If list is empty, then first and last are ignored.

If first is less than zero, it is considered to refer to before the first element of the list. For non-empty lists, the element indicated by first must exist or first must indicate before the start of the list.

If last is less than first, then any specified elements will be inserted into the list at the point specified by first with no elements being deleted.

The element arguments specify zero or more new arguments to be added to the list in place of those that were deleted. Each element argument will become a separate element of the list. If no element arguments are specified, then the elements between first and last are simply deleted. If list is empty, any element arguments are added to the end of the list.
EXAMPLES

Replacing an element of a list with another:

% lreplace \{a b c d e\} 1 1 foo
a foo c d e

Replacing two elements of a list with three:

% lreplace \{a b c d e\} 1 2 three more elements
a three more elements d e

Deleting the last element from a list in a variable:

% set var \{a b c d e\}
  a b c d e
% set var [lreplace $var end end]
  a b c d

A procedure to delete a given element from a list:

proc lremove \{listVariable value\} {
    upvar 1 \$listVariable var
    set idx [lsearch -exact \$var \$value]
    set var [lreplace \$var \$idx \$idx]
}

SEE ALSO

list, lappend, lindex, linsert, llength, lsearch, lset, lrange, lsort, string
KEYWORDS

element, list, replace

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 2001 Kevin B. Kenny <kennykb(at)acm.org>. All rights reserved.
NAME
registry - Manipulate the Windows registry

SYNOPSIS

registry broadcast keyName ?-timeout milliseconds?
registry delete keyName ?valueName?
registry get keyName valueName
registry keys keyName ?pattern?
registry set keyName ?valueName data ?type??
registry type keyName valueName
registry values keyName ?pattern?

SUPPORTED TYPES
binary
none
sz
expand_sz
dword
dword_big_endian
link
multi_sz
resource_list

PORTABILITY ISSUES

EXAMPLE

KEYWORDS

NAME
registry - Manipulate the Windows registry

SYNOPSIS

package require registry 1.1
registry option keyName ?arg arg ...?

DESCRIPTION

The registry package provides a general set of operations for manipulating the Windows registry. The package implements the registry Tcl command. This command is only supported on the Windows platform. Warning: this command should be used with caution as a corrupted registry can leave your system in an unusable state.

KeyName is the name of a registry key. Registry keys must be one of the following forms:

\hostname\rootname\keypath

rootname\keypath

rootname

Hostname specifies the name of any valid Windows host that exports its registry. The rootname component must be one of HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_CURRENT_CONFIG, HKEY_PERFORMANCE_DATA, or HKEY_DYN_DATA. The keypath can be one or more registry key names separated by backslash (\) characters.

Option indicates what to do with the registry key name. Any unique abbreviation for option is acceptable. The valid options are:

registry broadcast keyName ?-timeout milliseconds?
    Sends a broadcast message to the system and running programs to notify them of certain updates. This is necessary to propagate changes to key registry keys like Environment. The timeout specifies the amount of time, in milliseconds, to wait for applications to respond to the broadcast message. It defaults to 3000. The following example demonstrates how to add a path to the global Environment and notify applications of the change
without requiring a logoff/logon step (assumes admin privileges):

```plaintext
set regPath [join {
    HKEY_LOCAL_MACHINE
    SYSTEM
    CurrentControlSet
    Control
    {Session Manager}
    Environment
} "\"
set curPath [registry get $regPath "Path"]
registry set $regPath "Path" "$curPath;$addPath"
registry broadcast "Environment"
```

**registry delete** *keyName ?valueName?*

If the optional *valueName* argument is present, the specified value under *keyName* will be deleted from the registry. If the optional *valueName* is omitted, the specified key and any subkeys or values beneath it in the registry hierarchy will be deleted. If the key could not be deleted then an error is generated. If the key did not exist, the command has no effect.

**registry get** *keyName valueName*

Returns the data associated with the value *valueName* under the key *keyName*. If either the key or the value does not exist, then an error is generated. For more details on the format of the returned data, see **SUPPORTED TYPES**, below.

**registry keys** *keyName ?pattern?*

If *pattern* is not specified, returns a list of names of all the subkeys of *keyName*. If *pattern* is specified, only those names matching *pattern* are returned. Matching is determined using the same rules as for **string match**. If the specified *keyName* does not exist, then an error is generated.
registry set keyName ?valueName data ?type??
If valueName is not specified, creates the key keyName if it does not already exist. If valueName is specified, creates the key keyName and value valueName if necessary. The contents of valueName are set to data with the type indicated by type. If type is not specified, the type sz is assumed. For more details on the data and type arguments, see SUPPORTED TYPES below.

registry type keyName valueName
Returns the type of the value valueName in the key keyName. For more information on the possible types, see SUPPORTED TYPES, below.

registry values keyName ?pattern? 
If pattern is not specified, returns a list of names of all the values of keyName. If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for string match.

**SUPPORTED TYPES**

Each value under a key in the registry contains some data of a particular type in a type-specific representation. The registry command converts between this internal representation and one that can be manipulated by Tcl scripts. In most cases, the data is simply returned as a Tcl string. The type indicates the intended use for the data, but does not actually change the representation. For some types, the registry command returns the data in a different form to make it easier to manipulate. The following types are recognized by the registry command:

**binary**
The registry value contains arbitrary binary data. The data is represented exactly in Tcl, including any embedded nulls.

**none**
The registry value contains arbitrary binary data with no defined type. The data is represented exactly in Tcl, including any
embedded nulls.

sz
The registry value contains a null-terminated string. The data is represented in Tcl as a string.

expand_sz
The registry value contains a null-terminated string that contains unexpanded references to environment variables in the normal Windows style (for example, “%PATH%”). The data is represented in Tcl as a string.

dword
The registry value contains a little-endian 32-bit number. The data is represented in Tcl as a decimal string.

dword_big_endian
The registry value contains a big-endian 32-bit number. The data is represented in Tcl as a decimal string.

link
The registry value contains a symbolic link. The data is represented exactly in Tcl, including any embedded nulls.

multi_sz
The registry value contains an array of null-terminated strings. The data is represented in Tcl as a list of strings.

resource_list
The registry value contains a device-driver resource list. The data is represented exactly in Tcl, including any embedded nulls.

In addition to the symbolically named types listed above, unknown types are identified using a 32-bit integer that corresponds to the type code returned by the system interfaces. In this case, the data is represented exactly in Tcl, including any embedded nulls.

PORTABILITY ISSUES
The registry command is only available on Windows.

**EXAMPLE**

Print out how double-clicking on a Tcl script file will invoke a Tcl interpreter:

```tcl
package require registry
set ext .tcl

# Read the type name
set type [registry get HKEY_CLASSES_ROOT\$ext {}]
# Work out where to look for the command
set path HKEY_CLASSES_ROOT\$type\Shell\Open\command
# Read the command!
set command [registry get $path {}]

puts "$ext opens with $command"
```

**KEYWORDS**

registry

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997 Sun Microsystems, Inc.
Copyright © 2002 ActiveState Corporation.
NAME

tm - Facilities for locating and loading of Tcl Modules

SYNOPSIS

DESCRIPTION
::tcl::tm::path add path...
::tcl::tm::path remove path...
::tcl::tm::path list
::tcl::tm::roots path...

MODULE DEFINITION
FINDING MODULES
DEFAULT PATHS
SYSTEM SPECIFIC PATHS
   file normalize [info library]/../tclX/.
   file normalize EXEC/tclX/.
SITE SPECIFIC PATHS
   file normalize [info library]/../tclX/site-tcl
USER SPECIFIC PATHS
   $::env(TCLX_y_TM_PATH)
   $::env(TCLX_y_TM_PATH)

SEE ALSO

KEYWORDS

NAME

tm - Facilities for locating and loading of Tcl Modules

SYNOPSIS

::tcl::tm::path add path...
::tcl::tm::path remove path...
::tcl::tm::path list
::tcl::tm::roots path...
This document describes the facilities for locating and loading Tcl Modules. The following commands are supported:

```
::tcl::tm::path add path...
The paths are added at the head to the list of module paths, in order of appearance. This means that the last argument ends up as the new head of the list.

The command enforces the restriction that no path may be an ancestor directory of any other path on the list. If any of the new paths violates this restriction an error will be raised, before any of the paths have been added. In other words, if only one path argument violates the restriction then none will be added.

If a path is already present as is, no error will be raised and no action will be taken.

Paths are searched later in the order of their appearance in the list. As they are added to the front of the list they are searched in reverse order of addition. In other words, the paths added last are looked at first.

::tcl::tm::path remove path...
Removes the paths from the list of module paths. The command silently ignores all paths which are not on the list.

::tcl::tm::path list
Returns a list containing all registered module paths, in the order that they are searched for modules.

::tcl::tm::roots path...
Similar to path add, and layered on top of it. This command takes a list of paths, extends each with "tclX/site-tcl", and "tclXIX.y", for major version X of the Tcl interpreter and minor version y less than or equal to the minor version of the interpreter, and adds the resulting set of paths to the list of paths to search.
This command is used internally by the system to set up the system-specific default paths.

The command has been exposed to allow a build system to define additional root paths beyond those described by this document.

**MODULE DEFINITION**

A Tcl Module is a Tcl Package contained in a single file, and no other files required by it. This file has to be sourceable. In other words, a Tcl Module is always imported via:

```
source module_file
```

The `load` command is not directly used. This restriction is not an actual limitation, as some may believe. Ever since 8.4 the Tcl `source` command reads only until the first `^Z` character. This allows us to combine an arbitrary Tcl script with arbitrary binary data into one file, where the script processes the attached data in any it chooses to fully import and activate the package.

The name of a module file has to match the regular expression:

```
([[[:alpha:]][[:alnum:]]*)-([[[:digit:]].*)\.tm
```

The first capturing parentheses provides the name of the package, the second clause its version. In addition to matching the pattern, the extracted version number must not raise an error when used in the command:

```
package vcompare $version 0
```

**FINDING MODULES**
The directory tree for storing Tcl modules is separate from other parts of the filesystem and independent of auto_path.

Tcl Modules are searched for in all directories listed in the result of the command ::tcl::tm::path list. This is called the Module path. Neither the auto_path nor the tcl_pkgPath variables are used. All directories on the module path have to obey one restriction:

For any two directories, neither is an ancestor directory of the other.

This is required to avoid ambiguities in package naming. If for example the two directories “foo/” and “foo/cool” were on the path a package named cool::ice could be found via the names cool::ice or ice, the latter potentially obscuring a package named ice, unqualified.

Before the search is started, the name of the requested package is translated into a partial path, using the following algorithm:

All occurrences of “::” in the package name are replaced by the appropriate directory separator character for the platform we are on. On Unix, for example, this is “/”.

Example:

The requested package is encoding::base64. The generated partial path is “encoding/base64”.

After this translation the package is looked for in all module paths, by combining them one-by-one, first to last with the partial path to form a complete search pattern. Note that the search algorithm rejects all files where the filename does not match the regular expression given in the section MODULE DEFINITION. For the remaining files provide scripts are generated and added to the package if needed database.

The algorithm falls back to the previous unknown handler when none of the found module files satisfy the request. If the request was satisfied the fall-back is ignored.
Note that packages in module form have no control over the index and provide scripts entered into the package database for them. For a module file MF the index script is always:

```
package ifneeded PNAME PVERSION [list source MF]
```

and the provide script embedded in the above is:

```
source MF
```

Both package name PNAME and package version PVERSION are extracted from the filename MF according to the definition below:

```
MF = /module_path/PNAME'-PVERSION.tm
```

Where PNAME' is the partial path of the module as defined in section FINDING MODULES, and translated into PNAME by changing all directory separators to "::", and module_path is the path (from the list of paths to search) that we found the module file under.

Note also that we are here creating a connection between package names and paths. Tcl is case-sensitive when it comes to comparing package names, but there are filesystems which are not, like NTFS. Luckily these filesystems do store the case of the name, despite not using the information when comparing.

Given the above we allow the names for packages in Tcl modules to have mixed-case, but also require that there are no collisions when comparing names in a case-insensitive manner. In other words, if a package Foo is deployed in the form of a Tcl Module, packages like foo, fOo, etc. are not allowed anymore.

**DEFAULT PATHS**
The default list of paths on the module path is computed by a `tclsh` as follows, where \( X \) is the major version of the Tcl interpreter and \( y \) is less than or equal to the minor version of the Tcl interpreter.

All the default paths are added to the module path, even those paths which do not exist. Non-existent paths are filtered out during actual searches. This enables a user to create one of the paths searched when needed and all running applications will automatically pick up any modules placed in them.

The paths are added in the order as they are listed below, and for lists of paths defined by an environment variable in the order they are found in the variable.

### SYSTEM SPECIFIC PATHS

**file normalize [info library]/../tcl\( X \)/\( X \).\( y \)**

In other words, the interpreter will look into a directory specified by its major version and whose minor versions are less than or equal to the minor version of the interpreter.

For example for Tcl 8.4 the paths searched are:

```
[info library]/../tcl8/8.4
[info library]/../tcl8/8.3
[info library]/../tcl8/8.2
[info library]/../tcl8/8.1
[info library]/../tcl8/8.0
```

This definition assumes that a package defined for Tcl \( X.y \) can also be used by all interpreters which have the same major number \( X \) and a minor number greater than \( y \).

**file normalize EXEC/tcl\( X \)/\( X \).\( y \)**

Where `EXEC` is `file normalize [info nameofexecutable]/../lib` or `file normalize [::tcl::pkgconfig get libdir,runtime]`
This sets of paths is handled equivalently to the set coming before, except that it is anchored in EXEC_PREFIX. For a build with PREFIX = EXEC_PREFIX the two sets are identical.

### SITE SPECIFIC PATHS

**file normalize [info library]/../tclX/site-tcl**

Note that this is always a single entry because X is always a specific value (the current major version of Tcl).

### USER SPECIFIC PATHS

$::env(TCLX_y_TM_PATH)

A list of paths, separated by either : (Unix) or ; (Windows). This is user and site specific as this environment variable can be set not only by the user's profile, but by system configuration scripts as well.

$::env(TCLX.y_TM_PATH)

Same meaning and content as the previous variable. However the use of dot '.' to separate major and minor version number makes this name less to non-portable and its use is discouraged. Support of this variable has been kept only for backward compatibility with the original specification, i.e. TIP 189.

These paths are seen and therefore shared by all Tcl shells in the $::env(PATH) of the user.

Note that X and y follow the general rules set out above. In other words, Tcl 8.4, for example, will look at these 5 environment variables:

```
$::env(TCL8.4_TM_PATH)  $::env(TCL8_4_TM_PATH)
$::env(TCL8.3_TM_PATH)  $::env(TCL8_3_TM_PATH)
$::env(TCL8.2_TM_PATH)  $::env(TCL8_2_TM_PATH)
$::env(TCL8.1_TM_PATH)  $::env(TCL8_1_TM_PATH)
$::env(TCL8.0_TM_PATH)  $::env(TCL8_0_TM_PATH)
```
SEE ALSO

package, Tcl Improvement Proposal #189 “Tcl Modules” (online at http://tip.tcl.tk/189.html), Tcl Improvement Proposal #190 “Implementation Choices for Tcl Modules” (online at http://tip.tcl.tk/190.html)

KEYWORDS

modules, package

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2004-2008 Andreas Kupries <andreas_kupries(at)users.sourceforge.net>
NAME
filename - File name conventions supported by Tcl commands

INTRODUCTION

PATH TYPES

PATH SYNTAX
Unix

/ 
/etc/passwd
.
foo
foo/bar
../foo

Windows
\Host\share/file
c:foo
c:/foo
foo\bar
\foo
\\foo

TILDE SUBSTITUTION

PORTABILITY ISSUES

SEE ALSO

KEYWORDS

NAME

filename - File name conventions supported by Tcl commands

INTRODUCTION

All Tcl commands and C procedures that take file names as arguments expect the file names to be in one of three forms, depending on the
current platform. On each platform, Tcl supports file names in the standard forms(s) for that platform. In addition, on all platforms, Tcl supports a Unix-like syntax intended to provide a convenient way of constructing simple file names. However, scripts that are intended to be portable should not assume a particular form for file names. Instead, portable scripts must use the file split and file join commands to manipulate file names (see the file manual entry for more details).

### PATH TYPES

File names are grouped into three general types based on the starting point for the path used to specify the file: absolute, relative, and volume-relative. Absolute names are completely qualified, giving a path to the file relative to a particular volume and the root directory on that volume. Relative names are unqualified, giving a path to the file relative to the current working directory. Volume-relative names are partially qualified, either giving the path relative to the root directory on the current volume, or relative to the current directory of the specified volume. The file pathtype command can be used to determine the type of a given path.

### PATH SYNTAX

The rules for native names depend on the value reported in the Tcl array element tcl_platform(platform):

**Unix**

On Unix and Apple MacOS X platforms, Tcl uses path names where the components are separated by slashes. Path names may be relative or absolute, and file names may contain any character other than slash. The file names . and .. are special and refer to the current directory and the parent of the current directory respectively. Multiple adjacent slash characters are interpreted as a single separator. Any number of trailing slash characters at the end of a path are simply ignored, so the paths foo, fool and fool/ are all identical, and in particular fool does not necessarily mean a directory is being referred.
The following examples illustrate various forms of path names:

\/

Absolute path to the root directory.

/\etc/\passwd
Absolute path to the file named passwd in the directory etc in the root directory.

./
Relative path to the current directory.

foo
Relative path to the file foo in the current directory.

foo/bar
Relative path to the file bar in the directory foo in the current directory.

../foo
Relative path to the file foo in the directory above the current directory.

Windows
On Microsoft Windows platforms, Tcl supports both drive-relative and UNC style names. Both / and \ may be used as directory separators in either type of name. Drive-relative names consist of an optional drive specifier followed by an absolute or relative path. UNC paths follow the general form

\servername\sharename\path\file, but must at the very least contain the server and share components, i.e. \servername\sharename. In both forms, the file names . and .. are special and refer to the current directory and the parent of the current directory respectively. The following examples illustrate various forms of path names:

\\Host\share/file
Absolute UNC path to a file called file in the root directory of
the export point **share** on the host **Host**. Note that repeated use of **file dirname** on this path will give **//Host/share**, and will never give just **//Host**.

**c:foo**
Volume-relative path to a file **foo** in the current directory on drive **c**.

**c:/foo**
Absolute path to a file **foo** in the root directory of drive **c**.

**foo\bar**
Relative path to a file **bar** in the **foo** directory in the current directory on the current volume.

**\foo**
Volume-relative path to a file **foo** in the root directory of the current volume.

**\\foo**
Volume-relative path to a file **foo** in the root directory of the current volume. This is not a valid UNC path, so the assumption is that the extra backslashes are superfluous.

---

**TILDE SUBSTITUTION**

In addition to the file name rules described above, Tcl also supports **csh**-style tilde substitution. If a file name starts with a tilde, then the file name will be interpreted as if the first element is replaced with the location of the home directory for the given user. If the tilde is followed immediately by a separator, then the **$HOME** environment variable is substituted. Otherwise, the characters between the tilde and the next separator are taken as a user name, which is used to retrieve the user’s home directory for substitution. This works on Unix, MacOS X and Windows (except very old releases).

Old Windows platforms do not support tilde substitution when a user name follows the tilde. On these platforms, attempts to use a tilde
followed by a user name will generate an error that the user does not exist when Tcl attempts to interpret that part of the path or otherwise access the file. The behaviour of these paths when not trying to interpret them is the same as on Unix. File names that have a tilde without a user name will be correctly substituted using the $HOME environment variable, just like for Unix.

**PORTABILITY ISSUES**

Not all file systems are case sensitive, so scripts should avoid code that depends on the case of characters in a file name. In addition, the character sets allowed on different devices may differ, so scripts should choose file names that do not contain special characters like: <>:?"/|.

The safest approach is to use names consisting of alphanumeric characters only. Care should be taken with filenames which contain spaces (common on Windows systems) and filenames where the backslash is the directory separator (Windows native path names). Also Windows 3.1 only supports file names with a root of no more than 8 characters and an extension of no more than 3 characters.

On Windows platforms there are file and path length restrictions. Complete paths or filenames longer than about 260 characters will lead to errors in most file operations.

Another Windows peculiarity is that any number of trailing dots “.” in filenames are totally ignored, so, for example, attempts to create a file or directory with a name “foo.” will result in the creation of a file/directory with name “foo”. This fact is reflected in the results of `file normalize`. Furthermore, a file name consisting only of dots “..........” or dots with trailing characters “.....abc” is illegal.

**SEE ALSO**

file, glob

**KEYWORDS**

current directory, absolute file name, relative file name, volume-relative
file name, portability

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME

lreverse - Reverse the order of a list

SYNOPSIS

lreverse list

DESCRIPTION

The lreverse command returns a list that has the same elements as its input list, list, except with the elements in the reverse order.

EXAMPLES

lreverse {a a b c}
   → c b a a
lreverse {a b {c d} e f}
   → f e {c d} b a

SEE ALSO

list, lsearch, lsort

KEYWORDS

element, list, reverse
NAME
regsub - Perform substitutions based on regular expression pattern matching

SYNOPSIS
DESCRIPTION
-all
-expanded
-line
-linestop
-lineanchor
-nocase
-start index

EXAMPLES
SEE ALSO
KEYWORDS

NAME
regsub - Perform substitutions based on regular expression pattern matching

SYNOPSIS
regsub ?switches? exp string subSpec ?varName?

DESCRIPTION
This command matches the regular expression exp against string, and either copies string to the variable whose name is given by varName or returns string if varName is not present. (Regular expression matching is described in the re_syntax reference page.) If there is a match, then
while copying string to varName (or to the result of this command if varName is not present) the portion of string that matched exp is replaced with subSpec. If subSpec contains a “&” or “\0”, then it is replaced in the substitution with the portion of string that matched exp. If subSpec contains a “\n”, where n is a digit between 1 and 9, then it is replaced in the substitution with the portion of string that matched the n'th parenthesized subexpression of exp. Additional backslashes may be used in subSpec to prevent special interpretation of “&”, “\0”, “\n” and backslashes. The use of backslashes in subSpec tends to interact badly with the Tcl parser's use of backslashes, so it is generally safest to enclose subSpec in braces if it includes backslashes.

If the initial arguments to regsub start with - then they are treated as switches. The following switches are currently supported:

- all
  All ranges in string that match exp are found and substitution is performed for each of these ranges. Without this switch only the first matching range is found and substituted. If -all is specified, then “&” and “\n” sequences are handled for each substitution using the information from the corresponding match.

- expanded
  Enables use of the expanded regular expression syntax where whitespace and comments are ignored. This is the same as specifying the (?x) embedded option (see the re_syntax manual page).

- line
  Enables newline-sensitive matching. By default, newline is a completely ordinary character with no special meaning. With this flag, “[^\n]” bracket expressions and “.” never match newline, “^” matches an empty string after any newline in addition to its normal function, and “$” matches an empty string before any newline in addition to its normal function. This flag is equivalent to specifying both -linestop and -lineanchor, or the (?n) embedded option (see the re_syntax manual page).
-linestop
Changes the behavior of “[^]” bracket expressions and “.” so that they stop at newlines. This is the same as specifying the (?p) embedded option (see the re_syntax manual page).

-lineanchor
Changes the behavior of “^” and “$” (the “anchors”) so they match the beginning and end of a line respectively. This is the same as specifying the (?w) embedded option (see the re_syntax manual page).

-nocase
Upper-case characters in string will be converted to lower-case before matching against exp; however, substitutions specified by subSpec use the original unconverted form of string.

-start index
Specifies a character index offset into the string to start matching the regular expression at. The index value is interpreted in the same manner as the index argument to string_index. When using this switch, “^” will not match the beginning of the line, and \A will still match the start of the string at index. index will be constrained to the bounds of the input string.

--
Marks the end of switches. The argument following this one will be treated as exp even if it starts with a -.

If varName is supplied, the command returns a count of the number of matching ranges that were found and replaced, otherwise the string after replacement is returned. See the manual entry for regexp for details on the interpretation of regular expressions.

EXAMPLES

Replace (in the string in variable string) every instance of foo which is a word by itself with bar:
**re**

```bash
regsub -all {\mfoo\M} $string bar string
```

or (using the “basic regular expression” syntax):

```bash
regsub -all {(?b)\<foo\>} $string bar string
```

Insert double-quotes around the first instance of the word `interesting`, however it is capitalized.

```bash
regsub -nocase {\yinteresting\y} $string {"&"} string
```

Convert all non-ASCII and Tcl-significant characters into \u escape sequences by using `regsub` and `subst` in combination:

```bash
set RE {\[[{};#\\$/\0080-\uffff}\]}
set substitution {[format \\\\u%04x [scan "\\" %c]]}
set quoted [subst [regsub -all $RE $string $substitution]
```

**SEE ALSO**

`regexp`, `re_syntax`, `subst`, `string`

**KEYWORDS**

match, pattern, quoting, regular expression, substitute
NAME
trace - Monitor variable accesses, command usages and command executions

SYNOPSIS

DESCRIPTION

trace add type name ops ?args?
trace add command name ops commandPrefix
rename
delete
trace add execution name ops commandPrefix
enter
leave
enterstep
leavestep
trace add variable name ops commandPrefix
array
read
write
unset
trace remove type name opList commandPrefix
trace remove command name opList commandPrefix
trace remove execution name opList commandPrefix
trace remove variable name opList commandPrefix
trace info type name
trace info command name
trace info execution name
trace info variable name
trace variable name ops command
trace vdelete name ops command
trace vinfo name

EXAMPLES
trace - Monitor variable accesses, command usages and command executions

**SYNOPSIS**

```
trace option ?arg arg ...?
```

**DESCRIPTION**

This command causes Tcl commands to be executed whenever certain operations are invoked. The legal *options* (which may be abbreviated) are:

```
trace add type name ops ?args?
```

Where *type* is *command*, *execution*, or *variable*.

```
trace add command name ops commandPrefix
```

Arrange for *commandPrefix* to be executed (with additional arguments) whenever command *name* is modified in one of the ways given by the list *ops*. *Name* will be resolved using the usual namespace resolution rules used by commands. If the command does not exist, an error will be thrown.

*Ops* indicates which operations are of interest, and is a list of one or more of the following items:

- **rename**
  Invoke *commandPrefix* whenever the traced command is renamed. Note that renaming to the empty string is considered deletion, and will not be traced with "rename".

- **delete**
  Invoke *commandPrefix* when the traced command is deleted. Commands can be deleted explicitly by using the
rename command to rename the command to an empty string. Commands are also deleted when the interpreter is deleted, but traces will not be invoked because there is no interpreter in which to execute them.

When the trace triggers, depending on the operations being traced, a number of arguments are appended to commandPrefix so that the actual command is as follows:

```
commandPrefix oldName newName op
```

OldName and newName give the traced command's current (old) name, and the name to which it is being renamed (the empty string if this is a “delete” operation). Op indicates what operation is being performed on the command, and is one of rename or delete as defined above. The trace operation cannot be used to stop a command from being deleted. Tcl will always remove the command once the trace is complete. Recursive renaming or deleting will not cause further traces of the same type to be evaluated, so a delete trace which itself deletes the command, or a rename trace which itself renames the command will not cause further trace evaluations to occur. Both oldName and newName are fully qualified with any namespace(s) in which they appear.

**trace add execution name ops commandPrefix**

Arrange for commandPrefix to be executed (with additional arguments) whenever command name is executed, with traces occurring at the points indicated by the list ops. Name will be resolved using the usual namespace resolution rules used by commands. If the command does not exist, an error will be thrown.

Ops indicates which operations are of interest, and is a list of one or more of the following items:
**enter**
Invoke *commandPrefix* whenever the command *name* is executed, just before the actual execution takes place.

**leave**
Invoke *commandPrefix* whenever the command *name* is executed, just after the actual execution takes place.

**enterstep**
Invoke *commandPrefix* for every Tcl command which is executed from the start of the execution of the procedure *name* until that procedure finishes. *CommandPrefix* is invoked just before the actual execution of the Tcl command being reported takes place. For example if we have “proc foo {} { puts "hello" }”, then an *enterstep* trace would be invoked just before “puts "hello"” is executed. Setting an *enterstep* trace on a command *name* that does not refer to a procedure will not result in an error and is simply ignored.

**leavestep**
Invoke *commandPrefix* for every Tcl command which is executed from the start of the execution of the procedure *name* until that procedure finishes. *CommandPrefix* is invoked just after the actual execution of the Tcl command being reported takes place. Setting a *leavestep* trace on a command *name* that does not refer to a procedure will not result in an error and is simply ignored.

When the trace triggers, depending on the operations being traced, a number of arguments are appended to *commandPrefix* so that the actual command is as follows:

For *enter* and *enterstep* operations:

```
commandPrefix command-string op
```
Command-string gives the complete current command being executed (the traced command for a enter operation, an arbitrary command for a enterstep operation), including all arguments in their fully expanded form. Op indicates what operation is being performed on the command execution, and is one of enter or enterstep as defined above. The trace operation can be used to stop the command from executing, by deleting the command in question. Of course when the command is subsequently executed, an “invalid command” error will occur.

For leave and leavestep operations:

<table>
<thead>
<tr>
<th>command</th>
<th>command-string</th>
<th>code</th>
<th>result</th>
<th>op</th>
</tr>
</thead>
</table>

Command-string gives the complete current command being executed (the traced command for a enter operation, an arbitrary command for a enterstep operation), including all arguments in their fully expanded form. Code gives the result code of that execution, and result the result string. Op indicates what operation is being performed on the command execution, and is one of leave or leavestep as defined above. Note that the creation of many enterstep or leavestep traces can lead to unintuitive results, since the invoked commands from one trace can themselves lead to further command invocations for other traces.

CommandPrefix executes in the same context as the code that invoked the traced operation: thus the commandPrefix, if invoked from a procedure, will have access to the same local variables as code in the procedure. This context may be different than the context in which the trace was created. If commandPrefix invokes a procedure (which it normally does) then the procedure will have to use upvar or uplevel commands if it wishes to access the local variables of the code which invoked the trace operation.
While *commandPrefix* is executing during an execution trace, traces on *name* are temporarily disabled. This allows the *commandPrefix* to execute *name* in its body without invoking any other traces again. If an error occurs while executing the *commandPrefix*, then the command *name* as a whole will return that same error.

When multiple traces are set on *name*, then for *enter* and *enterstep* operations, the traced commands are invoked in the reverse order of how the traces were originally created; and for *leave* and *leavestep* operations, the traced commands are invoked in the original order of creation.

The behavior of execution traces is currently undefined for a command *name* imported into another namespace.

**trace add variable** *name* *ops* *commandPrefix*

Arrange for *commandPrefix* to be executed whenever variable *name* is accessed in one of the ways given by the list *ops*. *Name* may refer to a normal variable, an element of an array, or to an array as a whole (i.e. *name* may be just the name of an array, with no parenthesized index). If *name* refers to a whole array, then *commandPrefix* is invoked whenever any element of the array is manipulated. If the variable does not exist, it will be created but will not be given a value, so it will be visible to *namespace which* queries, but not to *info exists* queries.

*Ops* indicates which operations are of interest, and is a list of one or more of the following items:

**array**

Invoke *commandPrefix* whenever the variable is accessed or modified via the *array* command, provided that *name* is not a scalar variable at the time that the *array* command is invoked. If *name* is a scalar variable, the access via the *array* command will not trigger the trace.
read
Invoke \texttt{commandPrefix} whenever the variable is read.

write
Invoke \texttt{commandPrefix} whenever the variable is written.

unset
Invoke \texttt{commandPrefix} whenever the variable is unset. Variables can be unset explicitly with the \texttt{unset} command, or implicitly when procedures return (all of their local variables are unset). Variables are also unset when interpreters are deleted, but traces will not be invoked because there is no interpreter in which to execute them.

When the trace triggers, three arguments are appended to \texttt{commandPrefix} so that the actual command is as follows:

\begin{center}
\texttt{commandPrefix name1 name2 op}
\end{center}

\textit{Name1} and \textit{name2} give the name(s) for the variable being accessed: if the variable is a scalar then \textit{name1} gives the variable's name and \textit{name2} is an empty string; if the variable is an array element then \textit{name1} gives the name of the array and \textit{name2} gives the index into the array; if an entire array is being deleted and the trace was registered on the overall array, rather than a single element, then \textit{name1} gives the array name and \textit{name2} is an empty string. \textit{Name1} and \textit{name2} are not necessarily the same as the name used in the \texttt{trace variable} command: the \texttt{upvar} command allows a procedure to reference a variable under a different name. \textit{Op} indicates what operation is being performed on the variable, and is one of \texttt{read}, \texttt{write}, or \texttt{unset} as defined above.

\textit{CommandPrefix} executes in the same context as the code that invoked the traced operation: if the variable was accessed as part of a Tcl procedure, then \texttt{commandPrefix} will have access
to the same local variables as code in the procedure. This context may be different than the context in which the trace was created. If commandPrefix invokes a procedure (which it normally does) then the procedure will have to use upvar or uplevel if it wishes to access the traced variable. Note also that name1 may not necessarily be the same as the name used to set the trace on the variable; differences can occur if the access is made through a variable defined with the upvar command.

For read and write traces, commandPrefix can modify the variable to affect the result of the traced operation. If commandPrefix modifies the value of a variable during a read or write trace, then the new value will be returned as the result of the traced operation. The return value from commandPrefix is ignored except that if it returns an error of any sort then the traced operation also returns an error with the same error message returned by the trace command (this mechanism can be used to implement read-only variables, for example). For write traces, commandPrefix is invoked after the variable's value has been changed; it can write a new value into the variable to override the original value specified in the write operation. To implement read-only variables, commandPrefix will have to restore the old value of the variable.

While commandPrefix is executing during a read or write trace, traces on the variable are temporarily disabled. This means that reads and writes invoked by commandPrefix will occur directly, without invoking commandPrefix (or any other traces) again. However, if commandPrefix unsets the variable then unset traces will be invoked.

When an unset trace is invoked, the variable has already been deleted: it will appear to be undefined with no traces. If an unset occurs because of a procedure return, then the trace will be invoked in the variable context of the procedure being returned to: the stack frame of the returning procedure will no longer exist. Traces are not disabled during unset traces, so if
an unset trace command creates a new trace and accesses
the variable, the trace will be invoked. Any errors in unset
traces are ignored.

If there are multiple traces on a variable they are invoked in
order of creation, most-recent first. If one trace returns an
error, then no further traces are invoked for the variable. If an
array element has a trace set, and there is also a trace set on
the array as a whole, the trace on the overall array is invoked
before the one on the element.

Once created, the trace remains in effect either until the trace
is removed with the trace remove variable command
described below, until the variable is unset, or until the
interpreter is deleted. Unsetting an element of array will
remove any traces on that element, but will not remove traces
on the overall array.

This command returns an empty string.

trace remove type name opList commandPrefix
Where type is either command, execution or variable.

trace remove command name opList commandPrefix
If there is a trace set on command name with the operations
and command given by opList and commandPrefix, then the
trace is removed, so that commandPrefix will never again be
invoked. Returns an empty string. If name does not exist, the
command will throw an error.

trace remove execution name opList commandPrefix
If there is a trace set on command name with the operations
and command given by opList and commandPrefix, then the
trace is removed, so that commandPrefix will never again be
invoked. Returns an empty string. If name does not exist, the
command will throw an error.

trace remove variable name opList commandPrefix
If there is a trace set on variable $name$ with the operations and command given by $opList$ and $commandPrefix$, then the trace is removed, so that $commandPrefix$ will never again be invoked. Returns an empty string.

**trace info type name**
Where $type$ is either **command**, **execution** or **variable**.

**trace info command** $name$
Returns a list containing one element for each trace currently set on command $name$. Each element of the list is itself a list containing two elements, which are the $opList$ and $commandPrefix$ associated with the trace. If $name$ does not have any traces set, then the result of the command will be an empty string. If $name$ does not exist, the command will throw an error.

**trace info execution** $name$
Returns a list containing one element for each trace currently set on command $name$. Each element of the list is itself a list containing two elements, which are the $opList$ and $commandPrefix$ associated with the trace. If $name$ does not have any traces set, then the result of the command will be an empty string. If $name$ does not exist, the command will throw an error.

**trace info variable** $name$
Returns a list containing one element for each trace currently set on variable $name$. Each element of the list is itself a list containing two elements, which are the $opList$ and $commandPrefix$ associated with the trace. If $name$ does not exist or does not have any traces set, then the result of the command will be an empty string.

For backwards compatibility, three other subcommands are available:

**trace variable** $name$ $ops$ $command$
This is equivalent to **trace add variable** $name$ $ops$ $command$. 
trace vdelete name ops command
    This is equivalent to trace remove variable name ops command

trace vinfo name
    This is equivalent to trace info variable name

These subcommands are deprecated and will likely be removed in a future version of Tcl. They use an older syntax in which array, read, write, unset are replaced by a, r, w and u respectively, and the ops argument is not a list, but simply a string concatenation of the operations, such as rwua.

**EXAMPLES**

Print a message whenever either of the global variables foo and bar are updated, even if they have a different local name at the time (which can be done with the upvar command):

```
proc tracer {varname args} {
    upvar #0 $varname var
    puts "$varname was updated to be "$var"
}
trace add variable foo write "tracer foo"
trace add variable bar write "tracer bar"
```

Ensure that the global variable foobar always contains the product of the global variables foo and bar:

```
proc doMult args {
    global foo bar foobar
    set foobar [expr {$foo * $bar}]
}
trace add variable foo write doMult
trace add variable bar write doMult
```
Print a trace of what commands are executed during the processing of a Tcl procedure:

```
proc x {} { y }
proc y {} { z }
proc z {} { puts hello }
proc report args {puts [info level 0]}
trace add execution x enterstep report
x
  → report y enterstep
     report z enterstep
     report {puts hello} enterstep
     hello
```

SEE ALSO

`set`, `unset`

KEYWORDS

`read`, `command`, `rename`, `variable`, `write`, `trace`, `unset`
NAME

bgerror - Command invoked to process background errors

SYNOPSIS

bgerror message

DESCRIPTION

Release 8.5 of Tcl supports the `interp bgerror` command, which allows applications to register in an interpreter the command that will handle background errors in that interpreter. In older releases of Tcl, this level of control was not available, and applications could control the handling of background errors only by creating a command with the particular command name `bgerror` in the global namespace of an interpreter. The following documentation describes the interface requirements of the `bgerror` command an application might define to retain compatibility with pre-8.5 releases of Tcl. Applications intending to support only Tcl releases 8.5 and later should simply make use of `interp bgerror`.

The `bgerror` command does not exist as built-in part of Tcl. Instead, individual applications or users can define a `bgerror` command (e.g. as a Tcl procedure) if they wish to handle background errors.

A background error is one that occurs in an event handler or some other command that did not originate with the application. For example, if an error occurs while executing a command specified with the `after` command, then it is a background error. For a non-background error, the error can simply be returned up through nested Tcl command evaluations until it reaches the top-level code in the application; then the application can report the error in whatever way it wishes. When a
background error occurs, the unwinding ends in the Tcl library and there is no obvious way for Tcl to report the error.

When Tcl detects a background error, it saves information about the error and invokes a handler command registered by interp bgerror later as an idle event handler. The default handler command in turn calls the bgerror command. Before invoking bgerror, Tcl restores the errorInfo and errorCode variables to their values at the time the error occurred, then it invokes bgerror with the error message as its only argument. Tcl assumes that the application has implemented the bgerror command, and that the command will report the error in a way that makes sense for the application. Tcl will ignore any result returned by the bgerror command as long as no error is generated.

If another Tcl error occurs within the bgerror command (for example, because no bgerror command has been defined) then Tcl reports the error itself by writing a message to stderr.

If several background errors accumulate before bgerror is invoked to process them, bgerror will be invoked once for each error, in the order they occurred. However, if bgerror returns with a break exception, then any remaining errors are skipped without calling bgerror.

If you are writing code that will be used by others as part of a package or other kind of library, consider avoiding bgerror. The reason for this is that the application programmer may also want to define a bgerror, or use other code that does and thus will have trouble integrating your code.

### EXAMPLE

This bgerror procedure appends errors to a file, with a timestamp.

```tcl
proc bgerror {message} {
    set timestamp [clock format [clock seconds]]
    set fl [open mylog.txt {WRONLY CREATE APPEND}]
    puts $fl "$timestamp: bgerror in $::argv "$message"
    close $fl
}
```
close $f1
}

SEE ALSO

after, interp, tclvars

KEYWORDS

background, error, reporting

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
flush - Flush buffered output for a channel

flush channelId

Flushes any output that has been buffered for channelId.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdout or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension. The channel must have been opened for writing.

If the channel is in blocking mode the command does not return until all the buffered output has been flushed to the channel. If the channel is in nonblocking mode, the command may return before all buffered output has been flushed; the remainder will be flushed in the background as fast as the underlying file or device is able to absorb it.

Prompt for the user to type some information in on the console:

```
puts -nonewline "Please type your name: "
flush stdout
gets stdin name
puts "Hello there, $name!"
```
SEE ALSO

file, open, socket, Tcl_StandardChannels

KEYWORDS

blocking, buffer, channel, flush, nonblocking, output
NAME

lsearch - See if a list contains a particular element

SYNOPSIS

DESCRIPTION

MATCHING STYLE OPTIONS

-exact
-glob
-regexp
-sorted

GENERAL MODIFIER OPTIONS

-all
-inline
-not
-start index

CONTENTS DESCRIPTION OPTIONS

-ascii
-dictionary
-integer
-nocase
-real

SORTED LIST OPTIONS

-decreasing
-increasing

NESTED LIST OPTIONS

-index indexList
-subindices

EXAMPLES

SEE ALSO

KEYWORDS
Isearch - See if a list contains a particular element

**SYNOPSIS**

Isearch ?options? list pattern

**DESCRIPTION**

This command searches the elements of list to see if one of them matches pattern. If so, the command returns the index of the first matching element (unless the options -all or -inline are specified.) If not, the command returns -1. The option arguments indicates how the elements of the list are to be matched against pattern and must have one of the values below:

**MATCHING STYLE OPTIONS**

If all matching style options are omitted, the default matching style is -glob. If more than one matching style is specified, the last matching style given takes precedence.

- **exact**
  
  *Pattern* is a literal string that is compared for exact equality against each list element.

- **glob**

  *Pattern* is a glob-style pattern which is matched against each list element using the same rules as the string match command.

- **regexp**

  *Pattern* is treated as a regular expression and matched against each list element using the rules described in the re_syntax reference page.

- **sorted**

  The list elements are in sorted order. If this option is specified, Isearch will use a more efficient searching algorithm to search list. If no other options are specified, list is assumed to be sorted in
increasing order, and to contain ASCII strings. This option is mutually exclusive with `-glob` and `-regexp`, and is treated exactly like `-exact` when either `-all` or `-not` are specified.

**GENERAL MODIFIER OPTIONS**

These options may be given with all matching styles.

**-all**
Changes the result to be the list of all matching indices (or all matching values if `-inline` is specified as well.) If indices are returned, the indices will be in numeric order. If values are returned, the order of the values will be the order of those values within the input list.

**-inline**
The matching value is returned instead of its index (or an empty string if no value matches.) If `-all` is also specified, then the result of the command is the list of all values that matched.

**-not**
This negates the sense of the match, returning the index of the first non-matching value in the list.

**-start index**
The list is searched starting at position `index`. The interpretation of the `index` value is the same as for the command `string index`, supporting simple index arithmetic and indices relative to the end of the list.

**CONTENTS DESCRIPTION OPTIONS**

These options describe how to interpret the items in the list being searched. They are only meaningful when used with the `-exact` and `-sorted` options. If more than one is specified, the last one takes precedence. The default is `-ascii`.

**-ascii**
The list elements are to be examined as Unicode strings (the name is for backward-compatibility reasons.)

- **dictionary**
  The list elements are to be compared using dictionary-style comparisons (see `lsort` for a fuller description). Note that this only makes a meaningful difference from the `-ascii` option when the `-sorted` option is given, because values are only dictionary-equal when exactly equal.

- **integer**
  The list elements are to be compared as integers.

- **nocase**
  Causes comparisons to be handled in a case-insensitive manner. Has no effect if combined with the `-dictionary`, `-integer`, or `-real` options.

- **real**
  The list elements are to be compared as floating-point values.

### SORTED LIST OPTIONS

These options (only meaningful with the `-sorted` option) specify how the list is sorted. If more than one is given, the last one takes precedence. The default option is `-increasing`.

- **decreasing**
  The list elements are sorted in decreasing order. This option is only meaningful when used with `-sorted`.

- **increasing**
  The list elements are sorted in increasing order. This option is only meaningful when used with `-sorted`.

### NESTED LIST OPTIONS

These options are used to search lists of lists. They may be used with
any other options.

-index indexList
This option is designed for use when searching within nested lists. The indexList argument gives a path of indices (much as might be used with the lindex or lset commands) within each element to allow the location of the term being matched against.

-subindices
If this option is given, the index result from this command (or every index result when -all is also specified) will be a complete path (suitable for use with lindex or lset) within the overall list to the term found. This option has no effect unless the -index is also specified, and is just a convenience short-cut.

EXAMPLES

Basic searching:

```
lsearch {a b c d e} c
   → 2
lsearch -all {a b c a b c} c
   → 2 5
```

Using lsearch to filter lists:

```
lsearch -inline {a20 b35 c47} b*
   → b35
lsearch -inline -not {a20 b35 c47} b*
   → a20
lsearch -all -inline -not {a20 b35 c47} b*
   → a20 c47
lsearch -all -not {a20 b35 c47} b*
```
This can even do a “set-like” removal operation:

```
\texttt{lsearch} \ -all \ -inline \ -not \ -exact \ \{a \ b \ c \ a \ d \ e \ a \ f \ g\}
\rightarrow b \ c \ d \ e \ f \ g
```

Searching may start part-way through the list:

```
\texttt{lsearch} \ -start \ 3 \ \{a \ b \ c \ a \ b \ c\} \ c
\rightarrow 5
```

It is also possible to search inside elements:

```
\texttt{lsearch} \ -index \ 1 \ -all \ -inline \ \{\{a \ abc\} \ \{b \ bcd\} \ \{c \ cd\}\}
\rightarrow \{a \ abc\} \ \{b \ bcd\}
```

**SEE ALSO**

foreach, list, lappend, lindex, linsert, llength, lset, lsort, lrange, lreplace, string

**KEYWORDS**

list, match, pattern, regular expression, search, string
rename - Rename or delete a command

**SYNOPSIS**

rename oldName newName

**DESCRIPTION**

Rename the command that used to be called *oldName* so that it is now called *newName*. If *newName* is an empty string then *oldName* is deleted. *oldName* and *newName* may include namespace qualifiers (names of containing namespaces). If a command is renamed into a different namespace, future invocations of it will execute in the new namespace. The `rename` command returns an empty string as result.

**EXAMPLE**

The `rename` command can be used to wrap the standard Tcl commands with your own monitoring machinery. For example, you might wish to count how often the `source` command is called:

```tcl
rename ::source ::theRealSource
set sourceCount 0
proc ::source args {
    global sourceCount
    puts "called source for the [incr sourceCount]'th time"
    uplevel 1 ::theRealSource $args
}
```
SEE ALSO

namespace, proc

KEYWORDS

command, delete, namespace, rename

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
unknown - Handle attempts to use non-existent commands

unknown cmdName ?arg arg ...?

This command is invoked by the Tcl interpreter whenever a script tries to invoke a command that does not exist. The default implementation of unknown is a library procedure defined when Tcl initializes an interpreter. You can override the default unknown to change its functionality, or you can register a new handler for individual namespaces using the namespace unknown command. Note that there is no default implementation of unknown in a safe interpreter.

If the Tcl interpreter encounters a command name for which there is not a defined command (in either the current namespace, or the global namespace), then Tcl checks for the existence of an unknown handler for the current namespace. By default, this handler is a command named ::unknown. If there is no such command, then the interpreter returns an error. If the unknown command exists (or a new handler has been registered for the current namespace), then it is invoked with arguments consisting of the fully-substituted name and arguments for the original non-existent command. The unknown command typically does things like searching through library directories for a command procedure with the name cmdName, or expanding abbreviated command names to full-length, or automatically executing unknown commands as sub-processes. In some cases (such as expanding abbreviations) unknown will change the original command slightly and
then (re-)execute it. The result of the unknown command is used as the result for the original non-existent command.

The default implementation of unknown behaves as follows. It first calls the auto_load library procedure to load the command. If this succeeds, then it executes the original command with its original arguments. If the auto-load fails then unknown calls auto_execok to see if there is an executable file by the name cmd. If so, it invokes the Tcl exec command with cmd and all the args as arguments. If cmd cannot be auto-executed, unknown checks to see if the command was invoked at top-level and outside of any script. If so, then unknown takes two additional steps. First, it sees if cmd has one of the following three forms: !!, !event, or ^old^new?^. If so, then unknown carries out history substitution in the same way that csh would for these constructs. Finally, unknown checks to see if cmd is a unique abbreviation for an existing Tcl command. If so, it expands the command name and executes the command with the original arguments. If none of the above efforts has been able to execute the command, unknown generates an error return. If the global variable auto_noload is defined, then the auto-load step is skipped. If the global variable auto_noexec is defined then the auto-exec step is skipped. Under normal circumstances the return value from unknown is the return value from the command that was eventually executed.

**EXAMPLE**

Arrange for the unknown command to have its standard behavior except for first logging the fact that a command was not found:

```tcl
# Save the original one so we can chain to it
rename unknown _original_unknown

# Provide our own implementation
proc unknown args {
    puts stderr "WARNING: unknown command: $args"
    uplevel 1 [list _original_unknown {*}$args]
}
```
SEE ALSO

info, proc, interp, library, namespace

KEYWORDS

error, non-existent command

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
binary - Insert and extract fields from binary strings

SYNOPSIS

DESCRIPTION

BINARY FORMAT

a A
b B
H h
c c
s S
t t
i i
n n
w w
W W
m m
f f
r r
D D
Q q
x X
@ @

BINARY SCAN
PORTABILITY ISSUES
EXAMPLES
SEE ALSO
KEYWORDS

NAME

binary - Insert and extract fields from binary strings

SYNOPSIS

binary format formatString ?arg arg ...?
binary scan string formatString ?varName varName ...?
This command provides facilities for manipulating binary data. The first form, `binary format`, creates a binary string from normal Tcl values. For example, given the values 16 and 22, on a 32-bit architecture, it might produce an 8-byte binary string consisting of two 4-byte integers, one for each of the numbers. The second form of the command, `binary scan`, does the opposite: it extracts data from a binary string and returns it as ordinary Tcl string values.

**BINARY FORMAT**

The `binary format` command generates a binary string whose layout is specified by the `formatString` and whose contents come from the additional arguments. The resulting binary value is returned.

The `formatString` consists of a sequence of zero or more field specifiers separated by zero or more spaces. Each field specifier is a single type character followed by an optional flag character followed by an optional numeric `count`. Most field specifiers consume one argument to obtain the value to be formatted. The type character specifies how the value is to be formatted. The `count` typically indicates how many items of the specified type are taken from the value. If present, the `count` is a non-negative decimal integer or *, which normally indicates that all of the items in the value are to be used. If the number of arguments does not match the number of fields in the format string that consume arguments, then an error is generated. The flag character is ignored for `binary format`.

Here is a small example to clarify the relation between the field specifiers and the arguments:

```
  binary format d3d {1.0 2.0 3.0 4.0} 0.1
```

The first argument is a list of four numbers, but because of the count of
3 for the associated field specifier, only the first three will be used. The second argument is associated with the second field specifier. The resulting binary string contains the four numbers 1.0, 2.0, 3.0 and 0.1.

Each type-count pair moves an imaginary cursor through the binary data, storing bytes at the current position and advancing the cursor to just after the last byte stored. The cursor is initially at position 0 at the beginning of the data. The type may be any one of the following characters:

a

Stores a byte string of length count in the output string. Every character is taken as modulo 256 (i.e. the low byte of every character is used, and the high byte discarded) so when storing character strings not wholly expressible using the characters `\u0000-\u00ff`, the encoding convertto command should be used first to change the string into an external representation if this truncation is not desired (i.e. if the characters are not part of the ISO 8859-1 character set.) If arg has fewer than count bytes, then additional zero bytes are used to pad out the field. If arg is longer than the specified length, the extra characters will be ignored. If count is *, then all of the bytes in arg will be formatted. If count is omitted, then one character will be formatted. For example,

```
binary format a7a*a alpha bravo charlie
```

will return a string equivalent to `alpha\000\000bravoc`,

```
binary format a* [encoding convertto utf-8 \u20ac
```

will return a string equivalent to `\342\202\254` (which is the UTF-8 byte sequence for a Euro-currency character) and
A

This form is the same as `a` except that spaces are used for padding instead of nulls. For example,

```
binary format A6A*A alpha bravo charlie
```

will return `alpha bravoc`.

b

Stores a string of `count` binary digits in low-to-high order within each byte in the output string. `Arg` must contain a sequence of 1 and 0 characters. The resulting bytes are emitted in first to last order with the bits being formatted in low-to-high order within each byte. If `arg` has fewer than `count` digits, then zeros will be used for the remaining bits. If `arg` has more than the specified number of digits, the extra digits will be ignored. If `count` is *, then all of the digits in `arg` will be formatted. If `count` is omitted, then one digit will be formatted. If the number of bits formatted does not end at a byte boundary, the remaining bits of the last byte will be zeros. For example,
<table>
<thead>
<tr>
<th>Format</th>
<th>Binary String</th>
<th>Hex String</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td><code>b5b* 11100 111000011010</code></td>
<td><code>\x07\x87\x05</code></td>
</tr>
</tbody>
</table>

This form is the same as `b` except that the bits are stored in high-to-low order within each byte. For example,

<table>
<thead>
<tr>
<th>Format</th>
<th>Binary String</th>
<th>Hex String</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td><code>B5B* 11100 11100011010</code></td>
<td><code>\xe0\xe1\xa0</code></td>
</tr>
</tbody>
</table>

will return a string equivalent to `\xe0\xe1\xa0`.

**H**

Stores a string of `count` hexadecimal digits in high-to-low within each byte in the output string. `Arg` must contain a sequence of characters in the set “0123456789abcdefABCDEF”. The resulting bytes are emitted in first to last order with the hex digits being formatted in high-to-low order within each byte. If `arg` has fewer than `count` digits, then zeros will be used for the remaining digits. If `arg` has more than the specified number of digits, the extra digits will be ignored. If `count` is *, then all of the digits in `arg` will be formatted. If `count` is omitted, then one digit will be formatted. If the number of digits formatted does not end at a byte boundary, the remaining bits of the last byte will be zeros. For example,

<table>
<thead>
<tr>
<th>Format</th>
<th>Binary String</th>
<th>Hex String</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td><code>H3H*H2 ab DEF 987</code></td>
<td><code>\xab\x00\xde\xf0\x98</code></td>
</tr>
</tbody>
</table>

will return a string equivalent to `\xab\x00\xde\xf0\x98`.

**h**

This form is the same as `H` except that the digits are stored in low-to-high order within each byte. This is seldom required. For example,
will return a string equivalent to `\xba\x00\xed\x0f\x89`.

**c**

Stores one or more 8-bit integer values in the output string. If no `count` is specified, then `arg` must consist of an integer value. If `count` is specified, `arg` must consist of a list containing at least that many integers. The low-order 8 bits of each integer are stored as a one-byte value at the cursor position. If `count` is *, then all of the integers in the list are formatted. If the number of elements in the list is greater than `count`, then the extra elements are ignored. For example,

```
  binary format c3cc* {3 -3 128 1} 260 {2 5}
```

will return a string equivalent to `\x03\xfd\x80\x04\x02\x05`, whereas

```
  binary format c {2 5}
```

will generate an error.

**s**

This form is the same as **c** except that it stores one or more 16-bit integers in little-endian byte order in the output string. The low-order 16-bits of each integer are stored as a two-byte value at the cursor position with the least significant byte stored first. For example,

```
  binary format s3 {3 -3 258 1}
```
will return a string equivalent to $\texttt{\textbackslash x03\textbackslash x00\textbackslash xfd\textbackslash xff\textbackslash x02\textbackslash x01}$.

**S**

This form is the same as **s** except that it stores one or more 16-bit integers in big-endian byte order in the output string. For example,

```
| binary format | S3 {3 -3 258 1} |
```

will return a string equivalent to $\texttt{\textbackslash x00\textbackslash x03\textbackslash xff\textbackslash xfd\textbackslash x01\textbackslash x02}$.

**t**

This form (mnemonically **tiny**) is the same as **s** and **S** except that it stores the 16-bit integers in the output string in the native byte order of the machine where the Tcl script is running. To determine what the native byte order of the machine is, refer to the **byteOrder** element of the **tcl_platform** array.

**i**

This form is the same as **c** except that it stores one or more 32-bit integers in little-endian byte order in the output string. The low-order 32-bits of each integer are stored as a four-byte value at the cursor position with the least significant byte stored first. For example,

```
| binary format | i3 {3 -3 65536 1} |
```

will return a string equivalent to $\texttt{\textbackslash x03\textbackslash x00\textbackslash x00\textbackslash x00\textbackslash xfd\textbackslash xff\textbackslash xff\textbackslash xff\textbackslash x00\textbackslash x00\textbackslash x01\textbackslash x01\textbackslash x00}$.

**l**

This form is the same as **i** except that it stores one or more one or more 32-bit integers in big-endian byte order in the output string. For example,
The binary format I3 {3 -3 65536 1} will return a string equivalent to `\x00\x00\x00\x03\xff\xff\xff\xff\xff\xff\xff\xff\x00\x01\x00\x00`.

This form (mnemonically *number* or *normal*) is the same as i and l except that it stores the 32-bit integers in the output string in the native byte order of the machine where the Tcl script is running. To determine what the native byte order of the machine is, refer to the byteOrder element of the tcl_platform array.

The binary format w 7810179016327718216 will return the string **HelloTcl**.

This form is the same as c except that it stores one or more 64-bit integers in little-endian byte order in the output string. The low-order 64-bits of each integer are stored as an eight-byte value at the cursor position with the least significant byte stored first. For example,

The binary format w 7810179016327718216 will return the string **HelloTcl**.

This form is the same as w except that it stores one or more one or more 64-bit integers in big-endian byte order in the output string. For example,

The binary format Wc 4785469626960341345 110 will return the string **BigEndian**.
This form (mnemonically the mirror of w) is the same as w and W except that it stores the 64-bit integers in the output string in the native byte order of the machine where the Tcl script is running. To determine what the native byte order of the machine is, refer to the byteOrder element of the tcl_platform array.

f

This form is the same as c except that it stores one or more one or more single-precision floating point numbers in the machine's native representation in the output string. This representation is not portable across architectures, so it should not be used to communicate floating point numbers across the network. The size of a floating point number may vary across architectures, so the number of bytes that are generated may vary. If the value overflows the machine's native representation, then the value of FLT_MAX as defined by the system will be used instead. Because Tcl uses double-precision floating point numbers internally, there may be some loss of precision in the conversion to single-precision. For example, on a Windows system running on an Intel Pentium processor,

```
binary format f2 {1.6 3.4}
```

will return a string equivalent to \xcd\xcc\xcc\x3f\x9a\x99\x59\x40.

r

This form (mnemonically real) is the same as f except that it stores the single-precision floating point numbers in little-endian order. This conversion only produces meaningful output when used on machines which use the IEEE floating point representation (very common, but not universal.)

R

This form is the same as r except that it stores the single-precision floating point numbers in big-endian order.
This form is the same as f except that it stores one or more one or more double-precision floating point numbers in the machine's native representation in the output string. For example, on a Windows system running on an Intel Pentium processor,

\[
\text{\textbf{binary format}} \ d \ \{1.6\}
\]

will return a string equivalent to `\x9a\x99\x99\x99\x99\x99\xf9\x3f`.

This form (mnemonically the mirror of d) is the same as d except that it stores the double-precision floating point numbers in little-endian order. This conversion only produces meaningful output when used on machines which use the IEEE floating point representation (very common, but not universal.)

This form is the same as q except that it stores the double-precision floating point numbers in big-endian order.

Stores \textit{count} null bytes in the output string. If \textit{count} is not specified, stores one null byte. If \textit{count} is *, generates an error. This type does not consume an argument. For example,

\[
\text{\textbf{binary format}} \ a3\text{x}a3\times2a3 \ abc \ def \ ghi
\]

will return a string equivalent to `abc\x00\text{def}\x00\text{ghi}`.

Moves the cursor back \textit{count} bytes in the output string. If \textit{count} is * or is larger than the current cursor position, then the cursor is positioned at location 0 so that the next byte stored will be the first byte in the result string. If \textit{count} is omitted then the cursor is moved
back one byte. This type does not consume an argument. For example,

| **binary format** | a3X*a3X2a3 abc def ghi |

will return `dghi`.

@ Moves the cursor to the absolute location in the output string specified by `count`. Position 0 refers to the first byte in the output string. If `count` refers to a position beyond the last byte stored so far, then null bytes will be placed in the uninitialized locations and the cursor will be placed at the specified location. If `count` is *, then the cursor is moved to the current end of the output string. If `count` is omitted, then an error will be generated. This type does not consume an argument. For example,

| **binary format** | a5@2a1@*a3@10a1 abcde f ghi j |

will return `abfdeghi\000\000j`.

**BINARY SCAN**

The **binary scan** command parses fields from a binary string, returning the number of conversions performed. **String** gives the input bytes to be parsed (one byte per character, and characters not representable as a byte have their high bits chopped) and **formatString** indicates how to parse it. Each **varName** gives the name of a variable; when a field is scanned from **string** the result is assigned to the corresponding variable.

As with **binary format**, the **formatString** consists of a sequence of zero or more field specifiers separated by zero or more spaces. Each field specifier is a single type character followed by an optional flag character followed by an optional numeric `count`. Most field specifiers
consume one argument to obtain the variable into which the scanned values should be placed. The type character specifies how the binary data is to be interpreted. The count typically indicates how many items of the specified type are taken from the data. If present, the count is a non-negative decimal integer or *, which normally indicates that all of the remaining items in the data are to be used. If there are not enough bytes left after the current cursor position to satisfy the current field specifier, then the corresponding variable is left untouched and binary scan returns immediately with the number of variables that were set. If there are not enough arguments for all of the fields in the format string that consume arguments, then an error is generated. The flag character “u” may be given to cause some types to be read as unsigned values. The flag is accepted for all field types but is ignored for non-integer fields.

A similar example as with binary format should explain the relation between field specifiers and arguments in case of the binary scan subcommand:

```
binary scan $bytes s3s first second
```

This command (provided the binary string in the variable bytes is long enough) assigns a list of three integers to the variable first and assigns a single value to the variable second. If bytes contains fewer than 8 bytes (i.e. four 2-byte integers), no assignment to second will be made, and if bytes contains fewer than 6 bytes (i.e. three 2-byte integers), no assignment to first will be made. Hence:

```
puts [binary scan abcdefg s3s first second]
puts $first
puts $second
```

will print (assuming neither variable is set previously):
It is important to note that the c, s, and S (and i and I on 64bit systems) will be scanned into long data size values. In doing this, values that have their high bit set (0x80 for chars, 0x8000 for shorts, 0x80000000 for ints), will be sign extended. Thus the following will occur:

```
set signShort [binary format s1 0x8000]  
binary scan $signShort s1 val; # val == 0xFFFF8000
```

If you require unsigned values you can include the “u” flag character following the field type. For example, to read an unsigned short value:

```
set signShort [binary format s1 0x8000]  
binary scan $signShort su1 val; # val == 0x00008000
```

Each type-count pair moves an imaginary cursor through the binary data, reading bytes from the current position. The cursor is initially at position 0 at the beginning of the data. The type may be any one of the following characters:

- a

The data is a byte string of length `count`. If `count` is *, then all of the remaining bytes in `string` will be scanned into the variable. If `count` is omitted, then one byte will be scanned. All bytes scanned will be interpreted as being characters in the range \u0000-\u00ff so the `encoding convertfrom` command will be needed if the string is not a binary string or a string encoded in ISO 8859-1. For example,
**binary scan abcde\000fghi a6a10 var1 var2**

will return **1** with the string equivalent to **abcde\000** stored in **var1** and **var2** left unmodified, and

**binary scan \342\202\254 a* var1**
set var2 [encoding convertfrom utf-8 $var1]

will store a Euro-currency character in **var2**.

A

This form is the same as **a**, except trailing blanks and nulls are stripped from the scanned value before it is stored in the variable. For example,

**binary scan "abc efghi \000" A* var1**

will return **1** with **abc efghi** stored in **var1**.

b

The data is turned into a string of **count** binary digits in low-to-high order represented as a sequence of “1” and “0” characters. The data bytes are scanned in first to last order with the bits being taken in low-to-high order within each byte. Any extra bits in the last byte are ignored. If **count** is *, then all of the remaining bits in **string** will be scanned. If **count** is omitted, then one bit will be scanned. For example,

**binary scan \x07\x87\x05 b5b* var1 var2**

will return **2** with **11100** stored in **var1** and **1110000110100000** stored in **var2**.
This form is the same as **b**, except the bits are taken in high-to-low order within each byte. For example,

<table>
<thead>
<tr>
<th>Binary Scan</th>
<th>Format</th>
<th>var1</th>
<th>var2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>binary scan</code></td>
<td>\x70\x87\x05 B5B* var1 var2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...will return 2 with **01110** stored in `var1` and **100011100000101** stored in `var2`.

**H**

The data is turned into a string of `count` hexadecimal digits in high-to-low order represented as a sequence of characters in the set “0123456789abcdef”. The data bytes are scanned in first to last order with the hex digits being taken in high-to-low order within each byte. Any extra bits in the last byte are ignored. If `count` is *, then all of the remaining hex digits in `string` will be scanned. If `count` is omitted, then one hex digit will be scanned. For example,

<table>
<thead>
<tr>
<th>Binary Scan</th>
<th>Format</th>
<th>var1</th>
<th>var2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>binary scan</code></td>
<td>\x07\xC6\x05\x1f\x34 H3H* var1 var2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...will return 2 with **07c** stored in `var1` and **051f34** stored in `var2`.

**h**

This form is the same as **H**, except the digits are taken in reverse (low-to-high) order within each byte. For example,

<table>
<thead>
<tr>
<th>Binary Scan</th>
<th>Format</th>
<th>var1</th>
<th>var2</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>binary scan</code></td>
<td>\x07\x86\x05\x12\x34 h3h* var1 var2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...will return 2 with **706** stored in `var1` and **502143** stored in `var2`. Note that most code that wishes to parse the hexadecimal digits from multiple bytes in order should use the **H** format.
The data is turned into count 8-bit signed integers and stored in the corresponding variable as a list. If count is *, then all of the remaining bytes in string will be scanned. If count is omitted, then one 8-bit integer will be scanned. For example,

```
binary scan \x07\x86\x05 c2c* var1 var2
```

will return 2 with 7 -122 stored in var1 and 5 stored in var2. Note that the integers returned are signed, but they can be converted to unsigned 8-bit quantities using an expression like:

```
set num [expr \{ $num & 0xff \}]
```

The data is interpreted as count 16-bit signed integers represented in little-endian byte order. The integers are stored in the corresponding variable as a list. If count is *, then all of the remaining bytes in string will be scanned. If count is omitted, then one 16-bit integer will be scanned. For example,

```
binary scan \x05\x00\x07\x00\xf0\xff s2s* var1 var2
```

will return 2 with 5 7 stored in var1 and -16 stored in var2. Note that the integers returned are signed, but they can be converted to unsigned 16-bit quantities using an expression like:

```
set num [expr \{ $num & 0xffff \}]
```

This form is the same as s except that the data is interpreted as
*count* 16-bit signed integers represented in big-endian byte order. For example,

```
binary scan \x00\x05\x00\x07\xff\xf0  S2S* var1 var2
```

will return **2** with **5 7** stored in *var1* and **-16** stored in *var2*.

The data is interpreted as *count* 16-bit signed integers represented in the native byte order of the machine running the Tcl script. It is otherwise identical to **s** and **S**. To determine what the native byte order of the machine is, refer to the `byteOrder` element of the `tcl_platform` array.

The data is interpreted as *count* 32-bit signed integers represented in little-endian byte order. The integers are stored in the corresponding variable as a list. If *count* is *, then all of the remaining bytes in *string* will be scanned. If *count* is omitted, then one 32-bit integer will be scanned. For example,

```
set str \x05\x00\x00\x00\x07\x00\x00\x00\xf0\xff
binary scan $str i2i* var1 var2
```

will return **2** with **5 7** stored in *var1* and **-16** stored in *var2*. Note that the integers returned are signed, but they can be converted to unsigned 32-bit quantities using an expression like:

```
set num [expr { $num & 0xffffffff }]
```

This form is the same as **l** except that the data is interpreted as
count 32-bit signed integers represented in big-endian byte order. For example,

```
set str \x00\x00\x00\x05\x00\x00\x00\x07\xff\xff\xff
binary scan $str I2I* var1 var2
```

will return 2 with 5 7 stored in var1 and -16 stored in var2.

The data is interpreted as count 32-bit signed integers represented in the native byte order of the machine running the Tcl script. It is otherwise identical to i and I. To determine what the native byte order of the machine is, refer to the byteOrder element of the tcl_platform array.

w

The data is interpreted as count 64-bit signed integers represented in little-endian byte order. The integers are stored in the corresponding variable as a list. If count is *, then all of the remaining bytes in string will be scanned. If count is omitted, then one 64-bit integer will be scanned. For example,

```
set str \x05\x00\x00\x00\x07\x00\x00\x00\xf0\xff\xff\xff
binary scan $str wi* var1 var2
```

will return 2 with 30064771077 stored in var1 and -16 stored in var2. Note that the integers returned are signed and cannot be represented by Tcl as unsigned values.

W

This form is the same as w except that the data is interpreted as count 64-bit signed integers represented in big-endian byte order. For example,
```
set str \x00\x00\x00\x05\x00\x00\x07\xff\xff\n
binary scan $str W[*] var1 var2
```

will return 2 with **21474836487** stored in *var1* and **-16** stored in *var2*.

**m**

The data is interpreted as *count* 64-bit signed integers represented in the native byte order of the machine running the Tcl script. It is otherwise identical to *w* and *W*. To determine what the native byte order of the machine is, refer to the *byteOrder* element of the *tcl_platform* array.

**f**

The data is interpreted as *count* single-precision floating point numbers in the machine's native representation. The floating point numbers are stored in the corresponding variable as a list. If *count* is *, then all of the remaining bytes in *string* will be scanned. If *count* is omitted, then one single-precision floating point number will be scanned. The size of a floating point number may vary across architectures, so the number of bytes that are scanned may vary. If the data does not represent a valid floating point number, the resulting value is undefined and compiler dependent. For example, on a Windows system running on an Intel Pentium processor,

```
binary scan \x3f\xcc\xcc\xcd f var1
```

will return 1 with **1.6000000238418579** stored in *var1*.

**r**

This form is the same as *f* except that the data is interpreted as *count* single-precision floating point number in little-endian order. This conversion is not portable to the minority of systems not using...
IEEE floating point representations.

R

This form is the same as f except that the data is interpreted as count single-precision floating point number in big-endian order. This conversion is not portable to the minority of systems not using IEEE floating point representations.

d

This form is the same as f except that the data is interpreted as count double-precision floating point numbers in the machine's native representation. For example, on a Windows system running on an Intel Pentium processor,

```
  binary scan \x9a\x99\x99\x99\x99\x99\xf9\x3f
d  var1
```

will return 1 with 1.6000000000000001 stored in var1.

q

This form is the same as d except that the data is interpreted as count double-precision floating point number in little-endian order. This conversion is not portable to the minority of systems not using IEEE floating point representations.

Q

This form is the same as d except that the data is interpreted as count double-precision floating point number in big-endian order. This conversion is not portable to the minority of systems not using IEEE floating point representations.

x

Moves the cursor forward count bytes in string. If count is * or is larger than the number of bytes after the current cursor position, then the cursor is positioned after the last byte in string. If count is omitted, then the cursor is moved forward one byte. Note that this type does not consume an argument. For example,
binary scan \x01\x02\x03\x04 x2H* var1

will return 1 with 0304 stored in var1.

X

Moves the cursor back count bytes in string. If count is * or is larger than the current cursor position, then the cursor is positioned at location 0 so that the next byte scanned will be the first byte in string. If count is omitted then the cursor is moved back one byte. Note that this type does not consume an argument. For example,

binary scan \x01\x02\x03\x04 c2XH* var1 var2

will return 2 with 1 2 stored in var1 and 020304 stored in var2.

@

Moves the cursor to the absolute location in the data string specified by count. Note that position 0 refers to the first byte in string. If count refers to a position beyond the end of string, then the cursor is positioned after the last byte. If count is omitted, then an error will be generated. For example,

binary scan \x01\x02\x03\x04 c2@1H* var1 var2

will return 2 with 1 2 stored in var1 and 020304 stored in var2.

PORTABILITY ISSUES

The r, R, q and Q conversions will only work reliably for transferring data between computers which are all using IEEE floating point representations. This is very common, but not universal. To transfer floating-point numbers portably between all architectures, use their textual representation (as produced by format) instead.
EXAMPLES

This is a procedure to write a Tcl string to a binary-encoded channel as UTF-8 data preceded by a length word:

```tcl
proc writeString {channel string} {
    set data [encoding convertto utf-8 $string]
    puts -nonewline [binary format Ia* \ 
        [string length $data] $data]
}
```

This procedure reads a string from a channel that was written by the previously presented `writeString` procedure:

```tcl
proc readString {channel} {
    if {![binary scan [read $channel 4] I length]} {
        error "missing length"
    }
    set data [read $channel $length]
    return [encoding convertfrom utf-8 $data]
}
```

SEE ALSO

`format`, `scan`, `tclvars`

KEYWORDS

`binary`, `format`, `scan`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997 by Sun Microsystems, Inc.
NAME

for - 'For' loop

SYNOPSIS

for start test next body

DESCRIPTION

For is a looping command, similar in structure to the C for statement. The start, next, and body arguments must be Tcl command strings, and test is an expression string. The for command first invokes the Tcl interpreter to execute start. Then it repeatedly evaluates test as an expression; if the result is non-zero it invokes the Tcl interpreter on body, then invokes the Tcl interpreter on next, then repeats the loop. The command terminates when test evaluates to 0. If a continue command is invoked within body then any remaining commands in the current execution of body are skipped; processing continues by invoking the Tcl interpreter on next, then evaluating test, and so on. If a break command is invoked within body or next, then the for command will return immediately. The operation of break and continue are similar to the corresponding statements in C. For returns an empty string.

Note: test should almost always be enclosed in braces. If not, variable substitutions will be made before the for command starts executing, which means that variable changes made by the loop body will not be considered in the expression. This is likely to result in an infinite loop. If test is enclosed in braces, variable substitutions are delayed until the expression is evaluated (before each loop iteration), so changes in the variables will be visible. See below for an example:
EXAMPLES

Print a line for each of the integers from 0 to 10:

```
for {set x 0} {$x<10} {incr x} {
    puts "x is $x"
}
```

Either loop infinitely or not at all because the expression being evaluated is actually the constant, or even generate an error! The actual behaviour will depend on whether the variable \( x \) exists before the `for` command is run and whether its value is a value that is less than or greater than/equal to ten, and this is because the expression will be substituted before the `for` command is executed.

```
for {set x 0} {$x<10} {incr x} {
    puts "x is $x"
}
```

Print out the powers of two from 1 to 1024:

```
for {set x 1} {$x<=1024} {set x [expr {$x * 2}]} {
    puts "x is $x"
}
```

SEE ALSO

`break`, `continue`, `foreach`, `while`

KEYWORDS

`for`, `iteration`, `looping`
NAME

lset - Change an element in a list

SYNOPSIS

lset varName ?index...? newValue

DESCRIPTION

The lset command accepts a parameter, varName, which it interprets as the name of a variable containing a Tcl list. It also accepts zero or more indices into the list. The indices may be presented either consecutively on the command line, or grouped in a Tcl list and presented as a single argument. Finally, it accepts a new value for an element of varName.

If no indices are presented, the command takes the form:

\[\text{lset varName newValue}\]

or

\[\text{lset varName {} newValue}\]

In this case, newValue replaces the old value of the variable varName.

When presented with a single index, the lset command treats the content of the varName variable as a Tcl list. It addresses the index'th
element in it (0 refers to the first element of the list). When interpreting
the list, \texttt{lset} observes the same rules concerning braces and quotes
and backslashes as the Tcl command interpreter; however, variable
substitution and command substitution do not occur. The command
constructs a new list in which the designated element is replaced with
\texttt{newValue}. This new list is stored in the variable \texttt{varName}, and is also
the return value from the \texttt{lset} command.

If \textit{index} is negative or greater than or equal to the number of elements
in \texttt{$varName}, then an error occurs.

The interpretation of each simple \textit{index} value is the same as for the
command \texttt{string index}, supporting simple index arithmetic and indices
relative to the end of the list.

If additional \textit{index} arguments are supplied, then each argument is used
in turn to address an element within a sublist designated by the
previous indexing operation, allowing the script to alter elements in
sublists. The command,

\begin{verbatim}
 lset a 1 2 newValue
\end{verbatim}

or

\begin{verbatim}
 lset a \{1 2\} newValue
\end{verbatim}

replaces element 2 of sublist 1 with \texttt{newValue}.

The integer appearing in each \textit{index} argument must be greater than or
equal to zero. The integer appearing in each \textit{index} argument must be
strictly less than the length of the corresponding list. In other words, the
\texttt{lset} command cannot change the size of a list. If an index is outside the
permitted range, an error is reported.

\section*{EXAMPLES}
In each of these examples, the initial value of $x$ is:

```plaintext
set x [list [list a b c] [list d e f] [list g h i]]
→ {a b c} {d e f} {g h i}
```

The indicated return value also becomes the new value of $x$ (except in the last case, which is an error which leaves the value of $x$ unchanged.)

```plaintext
lset x {j k l}
→ j k l
lset x {} {j k l}
→ j k l
lset x 0 j
→ j {d e f} {g h i}
lset x 2 j
→ {a b c} {d e f} j
lset x end j
→ {a b c} {d e f} j
lset x end-1 j
→ {a b c} j {g h i}
lset x 2 1 j
→ {a b c} {d e f} {g j i}
lset x {2 1} j
→ {a b c} {d e f} {g j i}
lset x {2 3} j
→ list index out of range
```

In the following examples, the initial value of $x$ is:
The indicated return value also becomes the new value of \( x \).

```
\textbf{lset} \ x \ {1 \ 1 \ 0} \ j
\rightarrow \{{{a} \ {b}} \ {\{c} \ {d}\}} \ {{{e} \ {f}} \ {j} \ {h}}
```

```
\textbf{lset} \ x \ {1 \ 1 \ 0} \ j
\rightarrow \{{{a} \ {b}} \ {\{c} \ {d}\}} \ {{{e} \ {f}} \ {j} \ {h}}
```

**SEE ALSO**

list, lappend, lindex, linsert, llength, lsearch, lsort, lrange, lreplace, string

**KEYWORDS**

element, index, list, replace, set
return - Return from a procedure, or set return code of a script

SYNOPSIS

DESCRIPTION

EXCEPTIONAL RETURN CODES

ok (or 0)
error (1)
return (2)
break (3)
continue (4)
value

RETURN OPTIONS

-errorcode list
-errorinfo info
-level level
-options options

RETURN CODE HANDLING MECHANISMS

EXAMPLES

SEE ALSO

KEYWORDS
In its simplest usage, the `return` command is used without options in the body of a procedure to immediately return control to the caller of the procedure. If a `result` argument is provided, its value becomes the result of the procedure passed back to the caller. If `result` is not specified then an empty string will be returned to the caller as the result of the procedure.

The `return` command serves a similar function within script files that are evaluated by the `source` command. When `source` evaluates the contents of a file as a script, an invocation of the `return` command will cause script evaluation to immediately cease, and the value `result` (or an empty string) will be returned as the result of the `source` command.

**EXCEPTIONAL RETURN CODES**

In addition to the result of a procedure, the return code of a procedure may also be set by `return` through use of the `-code` option. In the usual case where the `-code` option is not specified the procedure will return normally. However, the `-code` option may be used to generate an exceptional return from the procedure. `Code` may have any of the following values:

- **ok (or 0)**
  Normal return: same as if the option is omitted. The return code of the procedure is 0 (`TCL_OK`).

- **error (1)**
  Error return: the return code of the procedure is 1 (`TCL_ERROR`). The procedure command behaves in its calling context as if it were the command `error result`. See below for additional options.

- **return (2)**
  The return code of the procedure is 2 (`TCL_RETURN`). The procedure command behaves in its calling context as if it were the command `return` (with no arguments).

- **break (3)**
  The return code of the procedure is 3 (`TCL_BREAK`). The
procedure command behaves in its calling context as if it were the command `break`.

**continue (4)**
The return code of the procedure is 4 (TCL_CONTINUE). The procedure command behaves in its calling context as if it were the command `continue`.

**value**
*Value* must be an integer; it will be returned as the return code for the current procedure.

When a procedure wants to signal that it has received invalid arguments from its caller, it may use `return -code error` with `result` set to a suitable error message. Otherwise usage of the `return -code` option is mostly limited to procedures that implement a new control structure.

The `return -code` command acts similarly within script files that are evaluated by the `source` command. During the evaluation of the contents of a file as a script by `source`, an invocation of the `return -code code` command will cause the return code of `source` to be `code`.

### RETURN OPTIONS

In addition to a result and a return code, evaluation of a command in Tcl also produces a dictionary of return options. In general usage, all `option value` pairs given as arguments to `return` become entries in the return options dictionary, and any values at all are acceptable except as noted below. The `catch` command may be used to capture all of this information — the return code, the result, and the return options dictionary — that arise from evaluation of a script.

As documented above, the `code` entry in the return options dictionary receives special treatment by Tcl. There are other return options also recognized and treated specially by Tcl. They are:

- `errorcode list`
The `-errorcode` option receives special treatment only when the value of the `-code` option is `TCL_ERROR`. Then the `list` value is meant to be additional information about the error, presented as a Tcl list for further processing by programs. If no `-errorcode` option is provided to `return` when the `-code error` option is provided, Tcl will set the value of the `-errorcode` entry in the return options dictionary to the default value of `NONE`. The `-errorcode` return option will also be stored in the global variable `errorCode`.

**-errorinfo info**
The `-errorinfo` option receives special treatment only when the value of the `-code` option is `TCL_ERROR`. Then `info` is the initial stack trace, meant to provide to a human reader additional information about the context in which the error occurred. The stack trace will also be stored in the global variable `errorInfo`. If no `-errorinfo` option is provided to `return` when the `-code error` option is provided, Tcl will provide its own initial stack trace value in the entry for `-errorinfo`. Tcl’s initial stack trace will include only the call to the procedure, and stack unwinding will append information about higher stack levels, but there will be no information about the context of the error within the procedure. Typically the `info` value is supplied from the value of `-errorinfo` in a return options dictionary captured by the `catch` command (or from the copy of that information stored in the global variable `errorInfo`).

**-level level**
The `-level` and `-code` options work together to set the return code to be returned by one of the commands currently being evaluated. The `level` value must be a non-negative integer representing a number of levels on the call stack. It defines the number of levels up the stack at which the return code of a command currently being evaluated should be `code`. If no `-level` option is provided, the default value of `level` is 1, so that `return` sets the return code that the current procedure returns to its caller, 1 level up the call stack. The mechanism by which these options work is described in more detail below.

**-options options**
The value options must be a valid dictionary. The entries of that dictionary are treated as additional option value pairs for the return command.

**RETURN CODE HANDLING MECHANISMS**

Return codes are used in Tcl to control program flow. A Tcl script is a sequence of Tcl commands. So long as each command evaluation returns a return code of **TCL_OK**, evaluation will continue to the next command in the script. Any exceptional return code (non-**TCL_OK**) returned by a command evaluation causes the flow on to the next command to be interrupted. Script evaluation ceases, and the exceptional return code from the command becomes the return code of the full script evaluation. This is the mechanism by which errors during script evaluation cause an interruption and unwinding of the call stack. It is also the mechanism by which commands like **break**, **continue**, and **return** cause script evaluation to terminate without evaluating all commands in sequence.

Some of Tcl's built-in commands evaluate scripts as part of their functioning. These commands can make use of exceptional return codes to enable special features. For example, the built-in Tcl commands that provide loops — such as **while**, **for**, and **foreach** — evaluate a script that is the body of the loop. If evaluation of the loop body returns the return code of **TCL_BREAK** or **TCL_CONTINUE**, the loop command can react in such a way as to give the **break** and **continue** commands their documented interpretation in loops.

Procedure invocation also involves evaluation of a script, the body of the procedure. Procedure invocation provides special treatment when evaluation of the procedure body returns the return code **TCL_RETURN**. In that circumstance, the **-level** entry in the return options dictionary is decremented. If after decrementing, the value of the **-level** entry is 0, then the value of the **-code** entry becomes the return code of the procedure. If after decrementing, the value of the **-level** entry is greater than zero, then the return code of the procedure is **TCL_RETURN**. If the procedure invocation occurred during the evaluation of the body of another procedure, the process will repeat
itself up the call stack, decrementing the value of the -level entry at each level, so that the code will be the return code of the current command level levels up the call stack. The source command performs the same handling of the TCL_RETURN return code, which explains the similarity of return invocation during a source to return invocation within a procedure.

The return code of the return command itself triggers this special handling by procedure invocation. If return is provided the option -level 0, then the return code of the return command itself will be the value code of the -code option (or TCL_OK by default). Any other value for the -level option (including the default value of 1) will cause the return code of the return command itself to be TCL_RETURN, triggering a return from the enclosing procedure.

EXAMPLES

First, a simple example of using return to return from a procedure, interrupting the procedure body.

```tcl
proc printOneLine {} {
    puts "line 1" ;# This line will be printed.
    return
    puts "line 2" ;# This line will not be printed
}
```

Next, an example of using return to set the value returned by the procedure.

```tcl
proc returnX {} {return X}
puts [returnX] ;# prints "X"
```

Next, a more complete example, using return -code error to report invalid arguments.

```tcl
```
proc factorial {n} {
    if {![string is integer $n] || ($n < 0)} {
        return -code error \
        "expected non-negative integer,\n        but got "$n""
    }
    if {$n < 2} {
        return 1
    }
    set m [expr {$n - 1}]
    set code [catch {factorial $m} factor]
    if {$code != 0} {
        return -code $code $factor
    }
    set product [expr {$n * $factor}]
    if {$product < 0} {
        return -code error \
        "overflow computing factorial of $n"
    }
    return $product
}

Next, a procedure replacement for break.

proc myBreak {} {
    return -code break
}

With the -level 0 option, return itself can serve as a replacement for break.

interp alias {} Break {} return -level 0 -code break
An example of using `catch` and `return -options` to re-raise a caught error:

```tcl
proc doSomething {} {
    set resource [allocate]
    catch {
        # Long script of operations
        # that might raise an error
    } result options
deallocate $resource
    return -options $options $result
}
```

Finally an example of advanced use of the `return` options to create a procedure replacement for `return` itself:

```tcl
proc myReturn {args} {
    set result ""
    if {([llength $args] % 2)} {
        set result [lindex $args end]
        set args [lrange $args 0 end-1]
    }
    set options [dict merge {-level 1} $args]
dict incr options -level
    return -options $options $result
}
```

**SEE ALSO**

break, `catch`, `continue`, `dict`, `error`, `proc`, `source`, `tclvars`

**KEYWORDS**

break, `catch`, `continue`, `error`, `procedure`, `return`
NAME
  unload - Unload machine code

SYNOPSIS

DESCRIPTION

- nocomplain
- keeplibrary

--

UNLOAD OPERATION

UNLOAD HOOK PROTOTYPE

NOTES

PORTABILITY ISSUES
  Unix

BUGS

EXAMPLE

SEE ALSO

KEYWORDS

NAME

unload - Unload machine code

SYNOPSIS

unload ?switches? fileName
unload ?switches? fileName packageName
unload ?switches? fileName packageName interp

DESCRIPTION

This command tries to unload shared libraries previously loaded with load from the application's address space. fileName is the name of the file containing the library file to be unload; it must be the same as the
filename provided to `load` for loading the library. The `packageName` argument is the name of the package (as determined by or passed to `load`), and is used to compute the name of the unload procedure; if not supplied, it is computed from `fileName` in the same manner as `load`. The `interp` argument is the path name of the interpreter from which to unload the package (see the `interp` manual entry for details); if `interp` is omitted, it defaults to the interpreter in which the `unload` command was invoked.

If the initial arguments to `unload` start with `-` then they are treated as switches. The following switches are currently supported:

- `nocomplain`  
  Suppresses all error messages. If this switch is given, `unload` will never report an error.

- `keeplibrary`  
  This switch will prevent `unload` from issuing the operating system call that will unload the library from the process.

--  
Marks the end of switches. The argument following this one will be treated as a `fileName` even if it starts with a `-`.

**UNLOAD OPERATION**

When a file containing a shared library is loaded through the `load` command, Tcl associates two reference counts to the library file. The first counter shows how many times the library has been loaded into normal (trusted) interpreters while the second describes how many times the library has been loaded into safe interpreters. As a file containing a shared library can be loaded only once by Tcl (with the first `load` call on the file), these counters track how many interpreters use the library. Each subsequent call to `load` after the first simply increments the proper reference count.

`unload` works in the opposite direction. As a first step, `unload` will check whether the library is unloadable: an unloadable library exports a
special unload procedure. The name of the unload procedure is
determined by **packageName** and whether or not the target interpreter
is a safe one. For normal interpreters the name of the initialization
procedure will have the form **pkg_Unload**, where **pkg** is the same as
**packageName** except that the first letter is converted to upper case and
all other letters are converted to lower case. For example, if
**packageName** is *foo* or *FOo*, the initialization procedure's name will be
**Foo_Unload**. If the target interpreter is a safe interpreter, then the
name of the initialization procedure will be **pkg_SafeUnload** instead of
**pkg_Unload**.

If **unload** determines that a library is not unloadable (or unload
functionality has been disabled during compilation), an error will be
returned. If the library is unloadable, then **unload** will call the unload
procedure. If the unload procedure returns **TCL_OK**, **unload** will
proceed and decrease the proper reference count (depending on the
target interpreter type). When both reference counts have reached 0,
the library will be detached from the process.

### UNLOAD HOOK PROTOTYPE

The unload procedure must match the following prototype:

```c
typedef int Tcl_PackageUnloadProc(Tcl_Interp *interp, int flags);
```

The **interp** argument identifies the interpreter from which the library is to
be unloaded. The unload procedure must return **TCL_OK** or
**TCL_ERROR** to indicate whether or not it completed successfully; in
the event of an error it should set the interpreter's result to point to an
error message. In this case, the result of the **unload** command will be
the result returned by the unload procedure.

The **flags** argument can be either
**TCL_UNLOAD_DETACH_FROM_INTERPRETER** or
**TCL_UNLOAD_DETACH_FROM_PROCESS**. In case the library will
remain attached to the process after the unload procedure returns (i.e. because the library is used by other interpreters),
TCL_UNLOAD_DETACH_FROM_INTERPRETER will be defined. However, if the library is used only by the target interpreter and the library will be detached from the application as soon as the unload procedure returns, the flags argument will be set to TCL_UNLOAD_DETACH_FROM_PROCESS.

NOTES

The unload command cannot unload libraries that are statically linked with the application. If fileName is an empty string, then the packageName argument must be specified.

If packageName is omitted or specified as an empty string, Tcl tries to guess the name of the package. This may be done differently on different platforms. The default guess, which is used on most UNIX platforms, is to take the last element of fileName, strip off the first three characters if they are lib, and use any following alphabetic and underline characters as the module name. For example, the command unload libxyz4.2.so uses the module name xyz and the command unload bin/last.so {} uses the module name last.

PORTABILITY ISSUES

Unix

Not all unix operating systems support library unloading. Under such an operating system unload returns an error (unless -nocomplain has been specified).

BUGS

If the same file is loaded by different fileNames, it will be loaded into the process's address space multiple times. The behavior of this varies from system to system (some systems may detect the redundant loads, others may not). In case a library has been silently detached by the operating system (and as a result Tcl thinks the library is still loaded), it may be dangerous to use unload on such a library (as the library will be
completely detached from the application while some interpreters will continue to use it).

### EXAMPLE

If an unloadable module in the file `foobar.dll` had been loaded using the `load` command like this (on Windows):

```bash
load c:/some/dir/foobar.dll
```

then it would be unloaded like this:

```bash
unload c:/some/dir/foobar.dll
```

This allows a C code module to be installed temporarily into a long-running Tcl program and then removed again (either because it is no longer needed or because it is being updated with a new version) without having to shut down the overall Tcl process.

### SEE ALSO

`info_sharedlibextension`, `load`, `safe`

### KEYWORDS

`binary code`, `unloading`, `safe interpreter`, `shared library`
NAME

break - Abort looping command

SYNOPSIS

break

DESCRIPTION

This command is typically invoked inside the body of a looping command such as for or foreach or while. It returns a TCL_BREAK code, which causes a break exception to occur. The exception causes the current script to be aborted out to the innermost containing loop command, which then aborts its execution and returns normally. Break exceptions are also handled in a few other situations, such as the catch command, Tk event bindings, and the outermost scripts of procedure bodies.

EXAMPLE

Print a line for each of the integers from 0 to 5:

```tcl
for {set x 0} {$x<10} {incr x} {
    if {$x > 5} {
        break
    }
    puts "x is $x"
}
```
SEE ALSO

catch, continue, for, foreach, return, while

KEYWORDS

abort, break, loop
NAME

foreach - Iterate over all elements in one or more lists

SYNOPSIS

foreach varname list body
foreach varlist1 list1 ?varlist2 list2 ...? body

DESCRIPTION

The `foreach` command implements a loop where the loop variable(s) take on values from one or more lists. In the simplest case there is one loop variable, `varname`, and one list, `list`, that is a list of values to assign to `varname`. The `body` argument is a Tcl script. For each element of `list` (in order from first to last), `foreach` assigns the contents of the element to `varname` as if the `lindex` command had been used to extract the element, then calls the Tcl interpreter to execute `body`.

In the general case there can be more than one value list (e.g., `list1` and `list2`), and each value list can be associated with a list of loop variables (e.g., `varlist1` and `varlist2`). During each iteration of the loop the variables of each `varlist` are assigned consecutive values from the corresponding `list`. Values in each `list` are used in order from first to last, and each value is used exactly once. The total number of loop iterations is large enough to use up all the values from all the value lists. If a value list does not contain enough elements for each of its loop variables in each iteration, empty values are used for the missing elements.

The `break` and `continue` statements may be invoked inside `body`, with the same effect as in the `for` command. `Foreach` returns an empty
string.

EXAMPLES

This loop prints every value in a list together with the square and cube of the value:

```tcl
set values {1 3 5 7 2 4 6 8} ;# Odd numbers first, for fun!
puts "Value\tSquare\t\tCube" ;# Neat-looking header
foreach x $values { ;# Now loop and print...
    puts "$x\t[expr \$x**2]\t\t[expr \$x**3]"
}
```

The following loop uses i and j as loop variables to iterate over pairs of elements of a single list.

```tcl
set x {}
foreach {i j} {a b c d e f} {
    lappend x $j $i
}
# The value of x is "b a d c f e"
# There are 3 iterations of the loop.
```

The next loop uses i and j to iterate over two lists in parallel.

```tcl
set x {}
foreach i {a b c} j {d e f g} {
    lappend x $i $j
}
# The value of x is "a d b e c f {} g"
# There are 4 iterations of the loop.
```
The two forms are combined in the following example.

```tcl
set x {}
foreach i {a b c} {j k} {d e f g} {
    lappend x $i $j $k
}
# The value of x is "a d e b f g c {} {}"
# There are 3 iterations of the loop.
```

SEE ALSO

for, while, break, continue

KEYWORDS

foreach, iteration, list, looping
NAME
Isort - Sort the elements of a list

SYNOPSIS

DESCRIPTION

-ascii
-dictionary
-integer
-real
-command command
-increasing
-decreasing
-indices
-index indexList
-nocase
-unique

NOTES

EXAMPLES

SEE ALSO

KEYWORDS

lsort - Sort the elements of a list

lsort ?options? list

This command sorts the elements of list, returning a new list in sorted order. The implementation of the Isort command uses the merge-sort
algorithm which is a stable sort that has $O(n \log n)$ performance characteristics.

By default ASCII sorting is used with the result returned in increasing order. However, any of the following options may be specified before list to control the sorting process (unique abbreviations are accepted):

- **-ascii**
  Use string comparison with Unicode code-point collation order (the name is for backward-compatibility reasons.) This is the default.

- **-dictionary**
  Use dictionary-style comparison. This is the same as **-ascii** except (a) case is ignored except as a tie-breaker and (b) if two strings contain embedded numbers, the numbers compare as integers, not characters. For example, in **-dictionary** mode, **bigBoy** sorts between **bigbang** and **bigboy**, and **x10y** sorts between **x9y** and **x11y**.

- **-integer**
  Convert list elements to integers and use integer comparison.

- **-real**
  Convert list elements to floating-point values and use floating comparison.

- **-command command**
  Use *command* as a comparison command. To compare two elements, evaluate a Tcl script consisting of *command* with the two elements appended as additional arguments. The script should return an integer less than, equal to, or greater than zero if the first element is to be considered less than, equal to, or greater than the second, respectively.

- **-increasing**
  Sort the list in increasing order ("smallest" items first). This is the default.
**-decreasing**
Sort the list in decreasing order ("largest" items first).

**-indices**
Return a list of indices into list in sorted order instead of the values themselves.

**-index indexList**
If this option is specified, each of the elements of list must itself be a proper Tcl sublist. Instead of sorting based on whole sublists, lsort will extract the indexList’th element from each sublist (as if the overall element and the indexList were passed to lindex) and sort based on the given element. For example,

```
lsort -integer -index 1
   {{First 24} {Second 18} {Third 30}}
```

returns **{Second 18} {First 24} {Third 30}**, and

```
lsort -index end-1
   {{a 1 e i} {b 2 3 f g} {c 4 5 6 d h}}
```

returns **{c 4 5 6 d h} {a 1 e i} {b 2 3 f g}**, and

```
lsort -index {0 1} {
   {{b i g} 12345}
   {{d e m o} 34512}
   {{c o d e} 54321}
}
```

returns **{{d e m o} 34512} {{b i g} 12345} {{c o d e} 54321}** (because e sorts before i which sorts before o.) This option is much more efficient than using **-command** to achieve the same effect.
-nocase
Causes comparisons to be handled in a case-insensitive manner. Has no effect if combined with the -dictionary, -integer, or -real options.

-unique
If this option is specified, then only the last set of duplicate elements found in the list will be retained. Note that duplicates are determined relative to the comparison used in the sort. Thus if -index 0 is used, \{1 a\} and \{1 b\} would be considered duplicates and only the second element, \{1 b\}, would be retained.

NOTES
The options to lsort only control what sort of comparison is used, and do not necessarily constrain what the values themselves actually are. This distinction is only noticeable when the list to be sorted has fewer than two elements.

The lsort command is reentrant, meaning it is safe to use as part of the implementation of a command used in the -command option.

EXAMPLES
Sorting a list using ASCII sorting:

% lsort {a10 B2 b1 a1 a2}
B2 a1 a10 a2 b1

Sorting a list using Dictionary sorting:

% lsort -dictionary {a10 B2 b1 a1 a2}
a1 a2 a10 b1 B2

Sorting lists of integers:
Sorting lists of floating-point numbers:

```
% lsort -real {5 3 1 2 11 4}
1 2 3 4 5 11
% lsort -real {0.5 0.07e1 0.4 6e-1}
0.4 0.5 6e-1 0.07e1
```

Sorting using indices:

```
% # Note the space character before the c
% lsort {{a 5} {c 3} {b 4} {e 1} {d 2}}
{c 3} {a 5} {b 4} {d 2} {e 1}
% lsort -index 0 {{a 5} {c 3} {b 4} {e 1} {d 2}}
{a 5} {b 4} {c 3} {d 2} {e 1}
% lsort -index 1 {{a 5} {c 3} {b 4} {e 1} {d 2}}
{e 1} {d 2} {c 3} {b 4} {a 5}
```

Stripping duplicate values using sorting:

```
% lsort -unique {a b c a b c a b c}
a b c
```

More complex sorting using a comparison function:

```
% proc compare {a b} {
```
set a0 [lindex $a 0]
set b0 [lindex $b 0]
if {$a0 < $b0} {
    return -1
} elseif {$a0 > $b0} {
    return 1
}
return [string compare [lindex $a 1] [lindex $b 1]]

% lsort -command compare \
    {{3 apple} {0x2 carrot} {1 dingo} {2 banana} \
     {1 dingo} {2 banana} {0x2 carrot} {3 apple}}

SEE ALSO

list, lappend, lindex, linsert, llength, lsearch, lset, lrange, lreplace

KEYWORDS

element, list, order, sort
NAME
Safe Base - A mechanism for creating and manipulating safe interpreters

SYNOPSIS

DESCRIPTION

COMMANDS

::safe::interpCreate ?slave? ?options...?
::safe::interpInit slave ?options...?
::safe::interpConfigure slave ?options...?
::safe::interpDelete slave
::safe::interpFindInAccessPath slave directory
::safe::interpAddToAccessPath slave directory
::safe::setLogCmd ?cmd arg...?

OPTIONS

-accessPath directoryList
-statics boolean
-noStatics
-nested boolean
-nestedLoadOk
-deleteHook script

ALIASES

source fileName
load fileName
file ?subCmd args...?
encoding ?subCmd args...?
exit

SECURITY
SEE ALSO
KEYWORDS
**NAME**

Safe Base - A mechanism for creating and manipulating safe interpreters

**SYNOPSIS**

::safe::interpCreate *slave* ?*options*?
::safe::interpInit *slave* ?*options*?
::safe::interpConfigure *slave* ?*options*?
::safe::interpDelete *slave*
::safe::interpAddToAccessPath *slave* *directory*
::safe::interpFindInAccessPath *slave* *directory*
::safe::setLogCmd *cmd* *arg*?

**OPTIONS**

?-accessPath *pathList*? ?-statics boolean? ?-noStatics? ?-nested boolean? ?-nestedLoadOk? ?-deleteHook *script*?

**DESCRIPTION**

Safe Tcl is a mechanism for executing untrusted Tcl scripts safely and for providing mediated access by such scripts to potentially dangerous functionality.

The Safe Base ensures that untrusted Tcl scripts cannot harm the hosting application. The Safe Base prevents integrity and privacy attacks. Untrusted Tcl scripts are prevented from corrupting the state of the hosting application or computer. Untrusted scripts are also prevented from disclosing information stored on the hosting computer or in the hosting application to any party.

The Safe Base allows a master interpreter to create safe, restricted interpreters that contain a set of predefined aliases for the `source`, `load`, `file`, `encoding`, and `exit` commands and are able to use the auto-loading and package mechanisms.
No knowledge of the file system structure is leaked to the safe interpreter, because it has access only to a virtualized path containing tokens. When the safe interpreter requests to source a file, it uses the token in the virtual path as part of the file name to source; the master interpreter transparently translates the token into a real directory name and executes the requested operation (see the section SECURITY below for details). Different levels of security can be selected by using the optional flags of the commands described below.

All commands provided in the master interpreter by the Safe Base reside in the safe namespace.

## COMMANDS

The following commands are provided in the master interpreter:

::safe::interpCreate ?slave? ?options...?
Creates a safe interpreter, installs the aliases described in the section ALIASES and initializes the auto-loading and package mechanism as specified by the supplied options. See the OPTIONS section below for a description of the optional arguments. If the slave argument is omitted, a name will be generated. ::safe::interpCreate always returns the interpreter name.

::safe::interpInit slave ?options...?
This command is similar to interpCreate except it that does not create the safe interpreter. slave must have been created by some other means, like interp create -safe.

::safe::interpConfigure slave ?options...?
If no options are given, returns the settings for all options for the named safe interpreter as a list of options and their current values for that slave. If a single additional argument is provided, it will return a list of 2 elements name and value where name is the full name of that option and value the current value for that option and the slave. If more than two additional arguments are provided, it will reconfigure the safe interpreter and change each and only the
provided options. See the section on OPTIONS below for options description. Example of use:

```
# Create new interp with the same configuration as $i0:
set i1 [safe::interpCreate {*}[safe::interpConfigure $i0]]

# Get the current deleteHook
set dh [safe::interpConfigure $i0 -del]

# Change (only) the statics loading ok attribute of interp and its deleteHook (leaving the rest unchanged):
safe::interpConfigure $i0 -delete {foo bar} -statics 0
```

::safe::interpDelete slave

Deletes the safe interpreter and cleans up the corresponding master interpreter data structures. If a deleteHook script was specified for this interpreter it is evaluated before the interpreter is deleted, with the name of the interpreter as an additional argument.

::safe::interpFindInAccessPath slave directory

This command finds and returns the token for the real directory directory in the safe interpreter's current virtual access path. It generates an error if the directory is not found. Example of use:

```
$slave eval [list set tk_library \  
    [::safe::interpFindInAccessPath $name $tk_]
```

::safe::interpAddToAccessPath slave directory

This command adds directory to the virtual path maintained for the safe interpreter in the master, and returns the token that can be used in the safe interpreter to obtain access to files in that directory. If the directory is already in the virtual path, it only returns the token without adding the directory to the virtual path again.
Example of use:

```
$slave eval [list set tk_library \\
    [:safe::interpAddToAccessPath $name $tk_library]
```

::safe::setLogCmd \?cmd arg...?\?
This command installs a script that will be called when interesting life cycle events occur for a safe interpreter. When called with no arguments, it returns the currently installed script. When called with one argument, an empty string, the currently installed script is removed and logging is turned off. The script will be invoked with one additional argument, a string describing the event of interest. The main purpose is to help in debugging safe interpreters. Using this facility you can get complete error messages while the safe interpreter gets only generic error messages. This prevents a safe interpreter from seeing messages about failures and other events that might contain sensitive information such as real directory names.

Example of use:

```
::safe::setLogCmd puts stderr
```

Below is the output of a sample session in which a safe interpreter attempted to source a file not found in its virtual access path. Note that the safe interpreter only received an error message saying that the file was not found:

```
NOTICE for slave interp10 : Created
NOTICE for slave interp10 : Setting accessPath=(/foo/bar)
NOTICE for slave interp10 : auto_path in interp10
ERROR for slave interp10 : /foo/bar/init.tcl: no
```
OPTIONS

The following options are common to ::safe::interpCreate, ::safe::interpInit, and ::safe::interpConfigure. Any option name can be abbreviated to its minimal non-ambiguous name. Option names are not case sensitive.

**-accessPath directoryList**
This option sets the list of directories from which the safe interpreter can source and load files. If this option is not specified, or if it is given as the empty list, the safe interpreter will use the same directories as its master for auto-loading. See the section SECURITY below for more detail about virtual paths, tokens and access control.

**-statics boolean**
This option specifies if the safe interpreter will be allowed to load statically linked packages (like load {} Tk). The default value is true: safe interpreters are allowed to load statically linked packages.

**-noStatics**
This option is a convenience shortcut for -statics false and thus specifies that the safe interpreter will not be allowed to load statically linked packages.

**-nested boolean**
This option specifies if the safe interpreter will be allowed to load packages into its own sub-interpreters. The default value is false: safe interpreters are not allowed to load packages into their own sub-interpreters.

**-nestedLoadOk**
This option is a convenience shortcut for -nested true and thus specifies the safe interpreter will be allowed to load packages into its own sub-interpreters.
**-deleteHook** *script*

When this option is given a non-empty *script*, it will be evaluated in
the master with the name of the safe interpreter as an additional
argument just before actually deleting the safe interpreter. Giving
an empty value removes any currently installed deletion hook script
for that safe interpreter. The default value ({}) is not to have any
deletion call back.

---

### ALIASES

The following aliases are provided in a safe interpreter:

**source** *fileName*

The requested file, a Tcl source file, is sourced into the safe
interpreter if it is found. The **source** alias can only source files from
directories in the virtual path for the safe interpreter. The **source**
alias requires the safe interpreter to use one of the token names in
its virtual path to denote the directory in which the file to be sourced
can be found. See the section on **SECURITY** for more discussion
of restrictions on valid filenames.

**load** *fileName*

The requested file, a shared object file, is dynamically loaded into
the safe interpreter if it is found. The filename must contain a token
name mentioned in the virtual path for the safe interpreter for it to
be found successfully. Additionally, the shared object file must
contain a safe entry point; see the manual page for the **load**
command for more details.

**file** *?subCmd args...?*

The **file** alias provides access to a safe subset of the
subcommands of the **file** command; it allows only **dirname**, **join**, **extension**, **root**, **tail**, **pathname** and **split** subcommands. For
more details on what these subcommands do see the manual page
for the **file** command.

**encoding** *?subCmd args...?*

The **encoding** alias provides access to a safe subset of the
subcommands of the **encoding** command; it disallows setting of the system encoding, but allows all other subcommands including **system** to check the current encoding.

**exit**

The calling interpreter is deleted and its computation is stopped, but the Tcl process in which this interpreter exists is not terminated.

### SECURITY

The Safe Base does not attempt to completely prevent annoyance and denial of service attacks. These forms of attack prevent the application or user from temporarily using the computer to perform useful work, for example by consuming all available CPU time or all available screen real estate. These attacks, while aggravating, are deemed to be of lesser importance in general than integrity and privacy attacks that the Safe Base is to prevent.

The commands available in a safe interpreter, in addition to the safe set as defined in **interp** manual page, are mediated aliases for **source**, **load**, **exit**, and safe subsets of **file** and **encoding**. The safe interpreter can also auto-load code and it can request that packages be loaded.

Because some of these commands access the local file system, there is a potential for information leakage about its directory structure. To prevent this, commands that take file names as arguments in a safe interpreter use tokens instead of the real directory names. These tokens are translated to the real directory name while a request to, e.g., source a file is mediated by the master interpreter. This virtual path system is maintained in the master interpreter for each safe interpreter created by ::safe::interpCreate or initialized by ::safe::interpInit and the path maps tokens accessible in the safe interpreter into real path names on the local file system thus preventing safe interpreters from gaining knowledge about the structure of the file system of the host on which the interpreter is executing. The only valid file names arguments for the **source** and **load** aliases provided to the slave are path in the form of [file join token filename] (i.e. when using the native file path formats: token\filename on Windows), where *token*
is representing one of the directories of the `accessPath` list and
`filename` is one file in that directory (no sub directories access are
allowed).

When a token is used in a safe interpreter in a request to source or load a
file, the token is checked and translated to a real path name and the file
to be sourced or loaded is located on the file system. The safe
interpreter never gains knowledge of the actual path name under which
the file is stored on the file system.

To further prevent potential information leakage from sensitive files that
are accidentally included in the set of files that can be sourced by a
safe interpreter, the `source` alias restricts access to files meeting the
following constraints: the file name must fourteen characters or shorter,
must not contain more than one dot ("."), must end up with the
extension (".tcl") or be called ("tclIndex").

Each element of the initial access path list will be assigned a token that
will be set in the slave `auto_path` and the first element of that list will be
set as the `tcl_library` for that slave.

If the access path argument is not given or is the empty list, the default
behavior is to let the slave access the same packages as the master
has access to (Or to be more precise: only packages written in Tcl
(which by definition cannot be dangerous as they run in the slave
interpreter) and C extensions that provides a _SafeInit entry point). For
that purpose, the master's `auto_path` will be used to construct the slave
access path. In order that the slave successfully loads the Tcl library
files (which includes the auto-loading mechanism itself) the `tcl_library`
will be added or moved to the first position if necessary, in the slave
access path, so the slave `tcl_library` will be the same as the master's
(its real path will still be invisible to the slave though). In order that auto-
loading works the same for the slave and the master in this by default
case, the first-level sub directories of each directory in the master
`auto_path` will also be added (if not already included) to the slave
access path. You can always specify a more restrictive path for which
sub directories will never be searched by explicitly specifying your
directory list with the `-accessPath` flag instead of relying on this default
mechanism.

When the `accessPath` is changed after the first creation or initialization (i.e. through `interpConfigure -accessPath list`), an `auto_reset` is automatically evaluated in the safe interpreter to synchronize its `auto_index` with the new token list.

**SEE ALSO**

`interp`, `library`, `load`, `package`, `source`, `unknown`

**KEYWORDS**

`alias`, `auto-loading`, `auto_mkindex`, `load`, `master interpreter`, `safe interpreter`, `slave interpreter`, `source`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME

unset - Delete variables

SYNOPSIS

unset ?-nocomplain? ?--? ?name name name ...?

DESCRIPTION

This command removes one or more variables. Each name is a variable name, specified in any of the ways acceptable to the set command. If a name refers to an element of an array then that element is removed without affecting the rest of the array. If a name consists of an array name with no parenthesized index, then the entire array is deleted. The unset command returns an empty string as result. If -nocomplain is specified as the first argument, any possible errors are suppressed. The option may not be abbreviated, in order to disambiguate it from possible variable names. The option -- indicates the end of the options, and should be used if you wish to remove a variable with the same name as any of the options. If an error occurs, any variables after the named one causing the error not deleted. An error can occur when the named variable does not exist, or the name refers to an array element but the variable is a scalar, or the name refers to a variable in a non-existent namespace.

EXAMPLE

Create an array containing a mapping from some numbers to their squares and remove the array elements for non-prime numbers:
array set squares {
    1 1 6 36
    2 4 7 49
    3 9 8 64
    4 16 9 81
    5 25 10 100
}

puts "The squares are:"
parray squares

unset squares(1) squares(4) squares(6)
unset squares(8) squares(9) squares(10)

puts "The prime squares are:"
parray squares

SEE ALSO
set, trace, upvar

KEYWORDS
remove, variable
NAME

catch - Evaluate script and trap exceptional returns

SYNOPSIS

catch script ?resultVarName? ?optionsVarName?

DESCRIPTION

The catch command may be used to prevent errors from aborting command interpretation. The catch command calls the Tcl interpreter recursively to execute script, and always returns without raising an error, regardless of any errors that might occur while executing script.

If script raises an error, catch will return a non-zero integer value corresponding to the exceptional return code returned by evaluation of script. Tcl defines the normal return code from script evaluation to be zero (0), or TCL_OK. Tcl also defines four exceptional return codes: 1 (TCL_ERROR), 2 (TCL_RETURN), 3 (TCL_BREAK), and 4 (TCL_CONTINUE). Errors during evaluation of a script are indicated by a return code of TCL_ERROR. The other exceptional return codes are returned by the return, break, and continue commands and in other special situations as documented. Tcl packages can define new commands that return other integer values as return codes as well, and scripts that make use of the return -code command can also have return codes other than the five defined by Tcl.

If the resultVarName argument is given, then the variable it names is set to the result of the script evaluation. When the return code from the script is 1 (TCL_ERROR), the value stored in resultVarName is an error message. When the return code from the script is 0 (TCL_OK), the
value stored in resultVarName is the value returned from script.

If the optionsVarName argument is given, then the variable it names is set to a dictionary of return options returned by evaluation of script. Tcl specifies two entries that are always defined in the dictionary: -code and -level. When the return code from evaluation of script is not TCL_RETURN, the value of the -level entry will be 0, and the value of the -code entry will be the same as the return code. Only when the return code is TCL_RETURN will the values of the -level and -code entries be something else, as further described in the documentation for the return command.

When the return code from evaluation of script is TCL_ERROR, three additional entries are defined in the dictionary of return options stored in optionsVarName: -errorinfo, -errorcode, and -errorline. The value of the -errorinfo entry is a formatted stack trace containing more information about the context in which the error happened. The formatted stack trace is meant to be read by a person. The value of the -errorcode entry is additional information about the error stored as a list. The -errorcode value is meant to be further processed by programs, and may not be particularly readable by people. The value of the -errorline entry is an integer indicating which line of script was being evaluated when the error occurred. The values of the -errorinfo and -errorcode entries of the most recent error are also available as values of the global variables ::errorInfo and ::errorCode respectively.

Tcl packages may provide commands that set other entries in the dictionary of return options, and the return command may be used by scripts to set return options in addition to those defined above.

**EXAMPLES**

The catch command may be used in an if to branch based on the success of a script.

```tcl
if { [catch {open $someFile w} fid] } {
    puts stderr "Could not open $someFile for writin"```
There are more complex examples of `catch` usage in the documentation for the `return` command.

**SEE ALSO**

`break`, `continue`, `dict`, `error`, `return`, `tclvars`

**KEYWORDS**

`catch`, `error`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
format - Format a string in the style of sprintf

SYNOPSIS

INTRODUCTION

DETAILS ON FORMATTING

- 
+ 
  space
0
#

d
u
i
0
x or X
c
s
f
  e or E
g or G
%

DIFFERENCES FROM ANSI SPRINTF

EXAMPLES

SEE ALSO

KEYWORDS
**format** `formatString arg arg ...`

---

**INTRODUCTION**

This command generates a formatted string in a fashion similar to the ANSI C `sprintf` procedure. `FormatString` indicates how to format the result, using `%` conversion specifiers as in `sprintf`, and the additional arguments, if any, provide values to be substituted into the result. The return value from `format` is the formatted string.

---

**DETAILS ON FORMATTING**

The command operates by scanning `formatString` from left to right. Each character from the format string is appended to the result string unless it is a percent sign. If the character is a `%` then it is not copied to the result string. Instead, the characters following the `%` character are treated as a conversion specifier. The conversion specifier controls the conversion of the next successive `arg` to a particular format and the result is appended to the result string in place of the conversion specifier. If there are multiple conversion specifiers in the format string, then each one controls the conversion of one additional `arg`. The `format` command must be given enough `args` to meet the needs of all of the conversion specifiers in `formatString`.

Each conversion specifier may contain up to six different parts: an XPG3 position specifier, a set of flags, a minimum field width, a precision, a size modifier, and a conversion character. Any of these fields may be omitted except for the conversion character. The fields that are present must appear in the order given above. The paragraphs below discuss each of these fields in turn.

If the `%` is followed by a decimal number and a $, as in “%2$d”, then the value to convert is not taken from the next sequential argument. Instead, it is taken from the argument indicated by the number, where 1 corresponds to the first `arg`. If the conversion specifier requires multiple arguments because of * characters in the specifier then successive arguments are used, starting with the argument given by the number. This follows the XPG3 conventions for positional specifiers. If there are
any positional specifiers in *formatString* then all of the specifiers must be positional.

The second portion of a conversion specifier may contain any of the following flag characters, in any order:

- Specifies that the converted argument should be left-justified in its field (numbers are normally right-justified with leading spaces if needed).

+ Specifies that a number should always be printed with a sign, even if positive.

*space*  
Specifies that a space should be added to the beginning of the number if the first character is not a sign.

0 Specifies that the number should be padded on the left with zeroes instead of spaces.

#  
Requests an alternate output form. For o and O conversions it guarantees that the first digit is always 0. For x or X conversions, ox or 0X (respectively) will be added to the beginning of the result unless it is zero. For all floating-point conversions (e, E, f, g, and G) it guarantees that the result always has a decimal point. For g and G conversions it specifies that trailing zeroes should not be removed.

The third portion of a conversion specifier is a decimal number giving a minimum field width for this conversion. It is typically used to make columns line up in tabular printouts. If the converted argument contains fewer characters than the minimum field width then it will be padded so that it is as wide as the minimum field width. Padding normally occurs by adding extra spaces on the left of the converted argument, but the 0
and - flags may be used to specify padding with zeroes on the left or with spaces on the right, respectively. If the minimum field width is specified as * rather than a number, then the next argument to the `format` command determines the minimum field width; it must be an integer value.

The fourth portion of a conversion specifier is a precision, which consists of a period followed by a number. The number is used in different ways for different conversions. For e, E, and f conversions it specifies the number of digits to appear to the right of the decimal point. For g and G conversions it specifies the total number of digits to appear, including those on both sides of the decimal point (however, trailing zeroes after the decimal point will still be omitted unless the # flag has been specified). For integer conversions, it specifies a minimum number of digits to print (leading zeroes will be added if necessary). For s conversions it specifies the maximum number of characters to be printed; if the string is longer than this then the trailing characters will be dropped. If the precision is specified with * rather than a number then the next argument to the `format` command determines the precision; it must be a numeric string.

The fifth part of a conversion specifier is a size modifier, which must be ll, h, or l. If it is ll it specifies that an integer value is taken without truncation for conversion to a formatted substring. If it is h it specifies that an integer value is truncated to a 16-bit range before converting. This option is rarely useful. If it is l it specifies that the integer value is truncated to the same range as that produced by the `wide()` function of the `expr` command (at least a 64-bit range). If neither h nor l are present, the integer value is truncated to the same range as that produced by the `int()` function of the `expr` command (at least a 32-bit range, but determined by the value of `tcl_platform(wordSize)`).

The last thing in a conversion specifier is an alphabetic character that determines what kind of conversion to perform. The following conversion characters are currently supported:

```
d
  Convert integer to signed decimal string.
```
Convert integer to unsigned decimal string.

Convert integer to signed decimal string (equivalent to d).

Convert integer to unsigned octal string.

Convert integer to unsigned hexadecimal string, using digits “0123456789abcdef” for x and “0123456789ABCDEF” for X).

Convert integer to the Unicode character it represents.

No conversion; just insert string.

Convert number to signed decimal string of the form xx.yyy, where the number of y’s is determined by the precision (default: 6). If the precision is 0 then no decimal point is output.

Convert number to scientific notation in the form x.yyyEE±zz, where the number of y’s is determined by the precision (default: 6). If the precision is 0 then no decimal point is output. If the E form is used then E is printed instead of e.

If the exponent is less than -4 or greater than or equal to the precision, then convert number as for %e or %E. Otherwise convert as for %f. Trailing zeroes and a trailing decimal point are omitted.

No conversion: just insert %.

DIFFERENCES FROM ANSI SPRINTF
The behavior of the format command is the same as the ANSI C `printf` procedure except for the following differences:

[1] %p and %n specifiers are not supported.

[2] For %c conversions the argument must be an integer value, which will then be converted to the corresponding character value.

[3] The size modifiers are ignored when formatting floating-point values. The ll modifier has no `printf` counterpart.

**EXAMPLES**

Convert the numeric value of a UNICODE character to the character itself:

```
set value 120
set char [format %c $value]
```

Convert the output of `time` into seconds to an accuracy of hundredths of a second:

```
set us [lindex [time $someTclCode] 0]
pus [format "%.2f seconds to execute" [expr {\$us / ...
```

Create a packed X11 literal color specification:

```
# Each color-component should be in range (0..255)
set color [format "#%02x%02x%02x" $r $g $b]
```
Use XPG3 format codes to allow reordering of fields (a technique that is often used in localized message catalogs; see `msgcat`) without reordering the data values passed to `format`:

```tcl
set fmt1 "Today, %d shares in %s were bought at $%.2f"
puts [format $fmt1 123 "Global BigCorp" 19.37]

set fmt2 "Bought %2\$s equity ($%3$.2f x %1\$d) today"
puts [format $fmt2 123 "Global BigCorp" 19.37]
```

Print a small table of powers of three:

```tcl
# Set up the column widths
set w1 5
set w2 10

# Make a nice header (with separator) for the table
set sep +-[string repeat - $w1]-+-[string repeat - $w2]-+
puts $sep
puts [format "| %-*s | %-*s |" $w1 "Index" $w2 "Power"]
puts $sep

# Print the contents of the table
set p 1
for {set i 0} {$i<=20} {incr i} {
    puts [format "| %*d | %*1d |" $w1 $i $w2 $p]
    set p [expr {wide($p) * 3}]
}

# Finish off by printing the separator again
puts $sep
```

**SEE ALSO**
scan, sprintf, string

KEYWORDS

conversion specifier, format, sprintf, string, substitution

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
mathfunc - Mathematical functions for Tcl expressions

SYNOPSIS

DESCRIPTION

DETAILED DEFINITIONS

abs arg
acos arg
asin arg
atan arg
atan2 y x
bool arg
ceil arg
cos arg
cosh arg
double arg
entier arg
exp arg
floor arg
fmod x y
hypot x y
int arg
isqrt arg
log arg
log10 arg
max arg ...
min arg ...
pow x y
rand
round arg
sin arg
sinh arg
NAME

mathfunc - Mathematical functions for Tcl expressions

SYNOPSIS

package require Tcl 8.5
::tcl::mathfunc::abs arg
::tcl::mathfunc::acos arg
::tcl::mathfunc::asin arg
::tcl::mathfunc::atan arg
::tcl::mathfunc::atan2 y x
::tcl::mathfunc::bool arg
::tcl::mathfunc::ceil arg
::tcl::mathfunc::cos arg
::tcl::mathfunc::cosh arg
::tcl::mathfunc::double arg
::tcl::mathfunc::entier arg
::tcl::mathfunc::exp arg
::tcl::mathfunc::floor arg
::tcl::mathfunc::fmod x y
::tcl::mathfunc::hypot x y
::tcl::mathfunc::int arg
::tcl::mathfunc::isqrt arg
::tcl::mathfunc::log arg
::tcl::mathfunc::log10 arg
::tcl::mathfunc::max arg ?arg ...?
::tcl::mathfunc::min arg ?arg ...?
::tcl::mathfunc::pow x y
::tcl::mathfunc::rand
::tcl::mathfunc::round arg
::tcl::mathfunc::sin arg
::tcl::mathfunc::sinh arg
::tcl::mathfunc::sqrt arg
::tcl::mathfunc::srand arg
::tcl::mathfunc::tan arg
::tcl::mathfunc::tanh arg
::tcl::mathfunc::wide arg

DESCRIPTION

The expr command handles mathematical functions of the form \textit{sin($x$)} or \textit{atan2($y,$x)} by converting them to calls of the form [\texttt{::tcl::mathfunc::sin [expr {$x}]}] or [\texttt{::tcl::mathfunc::atan2 [expr {$y}] [expr {$x}]}]. A number of math functions are available by default within the namespace \texttt{::tcl::mathfunc}; these functions are also available for code apart from expr, by invoking the given commands directly.

Tcl supports the following mathematical functions in expressions, all of which work solely with floating-point numbers unless otherwise noted:

\begin{verbatim}
  abs   acos   asin   atan
  atan2  bool  ceil  cos
  cosh  double  entier  exp
  floor  fmod   hypot  int
  isqrt  log    log10  max
  min   pow    rand   round
\end{verbatim}
In addition to these predefined functions, applications may define additional functions by using proc (or any other method, such as interp alias or Tcl_CreateObjCommand) to define new commands in the tcl::mathfunc namespace. In addition, an obsolete interface named Tcl_CreateMathFunc() is available to extensions that are written in C. The latter interface is not recommended for new implementations.

### DETAILED DEFINITIONS

**abs** *arg*

Returns the absolute value of *arg*. *Arg* may be either integer or floating-point, and the result is returned in the same form.

**acos** *arg*

Returns the arc cosine of *arg*, in the range \([0,\pi]\) radians. *Arg* should be in the range \([-1,1]\).

**asin** *arg*

Returns the arc sine of *arg*, in the range \([-\pi/2,\pi/2]\) radians. *Arg* should be in the range \([-1,1]\).

**atan** *arg*

Returns the arc tangent of *arg*, in the range \([-\pi/2,\pi/2]\) radians.

**atan2** *y* *x*

Returns the arc tangent of \(y/x\), in the range \([-\pi,\pi]\) radians. \(x\) and \(y\) cannot both be 0. If \(x\) is greater than 0, this is equivalent to \("atan [expr \(y/x\)]\"\).

**bool** *arg*

Accepts any numeric value, or any string acceptable to **string is**
**boolean**, and returns the corresponding boolean value 0 or 1. Non-zero numbers are true. Other numbers are false. Non-numeric strings produce boolean value in agreement with **string is true** and **string is false**.

**ceil** arg
Returns the smallest integral floating-point value (i.e. with a zero fractional part) not less than arg. The argument may be any numeric value.

**cos** arg
Returns the cosine of arg, measured in radians.

**cosh** arg
Returns the hyperbolic cosine of arg. If the result would cause an overflow, an error is returned.

**double** arg
The argument may be any numeric value. If arg is a floating-point value, returns arg, otherwise converts arg to floating-point and returns the converted value. May return Inf or -Inf when the argument is a numeric value that exceeds the floating-point range.

**entier** arg
The argument may be any numeric value. The integer part of arg is determined and returned. The integer range returned by this function is unlimited, unlike int and wide which truncate their range to fit in particular storage widths.

**exp** arg
Returns the exponential of arg, defined as e**arg. If the result would cause an overflow, an error is returned.

**floor** arg
Returns the largest integral floating-point value (i.e. with a zero fractional part) not greater than arg. The argument may be any numeric value.
fmod $x$ $y$
Returns the floating-point remainder of the division of $x$ by $y$. If $y$ is 0, an error is returned.

hypot $x$ $y$
Computes the length of the hypotenuse of a right-angled triangle “sqrt [expr \{x*x+y*y\}].”

int $arg$
The argument may be any numeric value. The integer part of $arg$ is determined, and then the low order bits of that integer value up to the machine word size are returned as an integer value. For reference, the number of bytes in the machine word are stored in tcl_platform(wordSize).

isqrt $arg$
Computes the integer part of the square root of $arg$. Arg must be a positive value, either an integer or a floating point number. Unlike sqrt, which is limited to the precision of a floating point number, isqrt will return a result of arbitrary precision.

log $arg$
Returns the natural logarithm of $arg$. Arg must be a positive value.

log10 $arg$
Returns the base 10 logarithm of $arg$. Arg must be a positive value.

max $arg$ ...
Accepts one or more numeric arguments. Returns the one argument with the greatest value.

min $arg$ ...
Accepts one or more numeric arguments. Returns the one argument with the least value.

pow $x$ $y$
Computes the value of $x$ raised to the power $y$. If $x$ is negative, $y$ must be an integer value.
**rand**

Returns a pseudo-random floating-point value in the range \((0,1)\). The generator algorithm is a simple linear congruential generator that is not cryptographically secure. Each result from `rand` completely determines all future results from subsequent calls to `rand`, so `rand` should not be used to generate a sequence of secrets, such as one-time passwords. The seed of the generator is initialized from the internal clock of the machine or may be set with the `srand` function.

**round arg**

If `arg` is an integer value, returns `arg`, otherwise converts `arg` to integer by rounding and returns the converted value.

**sin arg**

Returns the sine of `arg`, measured in radians.

**sinh arg**

Returns the hyperbolic sine of `arg`. If the result would cause an overflow, an error is returned.

**sqrt arg**

The argument may be any non-negative numeric value. Returns a floating-point value that is the square root of `arg`. May return `Inf` when the argument is a numeric value that exceeds the square of the maximum value of the floating-point range.

**srand arg**

The `arg`, which must be an integer, is used to reset the seed for the random number generator of `rand`. Returns the first random number (see `rand`) from that seed. Each interpreter has its own seed.

**tan arg**

Returns the tangent of `arg`, measured in radians.

**tanh arg**

Returns the hyperbolic tangent of `arg`. 
wide arg

The argument may be any numeric value. The integer part of arg is determined, and then the low order 64 bits of that integer value are returned as an integer value.

SEE ALSO

expr, mathop, namespace

COPYRIGHT

Copyright (c) 1993 The Regents of the University of California.
Copyright (c) 1994-2000 Sun Microsystems Incorporated.
Copyright (c) 2005, 2006 by Kevin B. Kenny <kennykb@acm.org>.

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-2000 Sun Microsystems, Inc.
Copyright © 2005 by Kevin B. Kenny <kennykb(at)acm.org>. All rights reserved
NAME

scan - Parse string using conversion specifiers in the style of sscanf

SYNOPSIS

scan string format ?varName varName ...?

INTRODUCTION
This command parses substrings from an input string in a fashion similar to the ANSI C `sscanf` procedure and returns a count of the number of conversions performed, or -1 if the end of the input string is reached before any conversions have been performed. *String* gives the input to be parsed and *format* indicates how to parse it, using % conversion specifiers as in `sscanf`. Each *varName* gives the name of a variable; when a substring is scanned from *string* that matches a conversion specifier, the substring is assigned to the corresponding variable. If no *varName* variables are specified, then **scan** works in an inline manner, returning the data that would otherwise be stored in the variables as a list. In the inline case, an empty string is returned when the end of the input string is reached before any conversions have been performed.

### DETAILS ON SCANNING

**Scan** operates by scanning *string* and *format* together. If the next character in *format* is a blank or tab then it matches any number of white space characters in *string* (including zero). Otherwise, if it is not a % character then it must match the next character of *string*. When a % is encountered in *format*, it indicates the start of a conversion specifier. A conversion specifier contains up to four fields after the %: a XPG3 position specifier (or a * to indicate the converted value is to be discarded instead of assigned to any variable); a number indicating a maximum substring width; a size modifier; and a conversion character. All of these fields are optional except for the conversion character. The fields that are present must appear in the order given above.

When **scan** finds a conversion specifier in *format*, it first skips any white-space characters in *string* (unless the conversion character is [ or c). Then it converts the next input characters according to the conversion specifier and stores the result in the variable given by the next argument to **scan**.

If the % is followed by a decimal number and a $, as in “%2$d”, then the variable to use is not taken from the next sequential argument. Instead, it is taken from the argument indicated by the number, where 1 corresponds to the first *varName*. If there are any positional specifiers
in *format* then all of the specifiers must be positional. Every *varName* on the argument list must correspond to exactly one conversion specifier or an error is generated, or in the inline case, any position can be specified at most once and the empty positions will be filled in with empty strings.

The size modifier field is used only when scanning a substring into one of Tcl's integer values. The size modifier field dictates the integer range acceptable to be stored in a variable, or, for the inline case, in a position in the result list. The syntactically valid values for the size modifier are *h*, *L*, *I*, and *ll*. The *h* size modifier value is equivalent to the absence of a size modifier in the the conversion specifier. Either one indicates the integer range to be stored is limited to the same range produced by the `int()` function of the `expr` command. The *L* size modifier is equivalent to the *I* size modifier. Either one indicates the integer range to be stored is limited to the same range produced by the `wide()` function of the `expr` command. The *ll* size modifier indicates that the integer range to be stored is unlimited.

The following conversion characters are supported:

*d*

The input substring must be a decimal integer. It is read in and the integer value is stored in the variable, truncated as required by the size modifier value.

*o*

The input substring must be an octal integer. It is read in and the integer value is stored in the variable, truncated as required by the size modifier value.

*x*

The input substring must be a hexadecimal integer. It is read in and the integer value is stored in the variable, truncated as required by the size modifier value.

*u*

The input substring must be a decimal integer. The integer value is
truncated as required by the size modifier value, and the corresponding unsigned value for that truncated range is computed and stored in the variable as a decimal string. The conversion makes no sense without reference to a truncation range, so the size modifier \texttt{ll} is not permitted in combination with conversion character \texttt{u}.

\texttt{i}

The input substring must be an integer. The base (i.e. decimal, binary, octal, or hexadecimal) is determined in the same fashion as described in \texttt{expr}. The integer value is stored in the variable, truncated as required by the size modifier value.

\texttt{c}

A single character is read in and its Unicode value is stored in the variable as an integer value. Initial white space is not skipped in this case, so the input substring may be a white-space character.

\texttt{s}

The input substring consists of all the characters up to the next white-space character; the characters are copied to the variable.

\texttt{e} or \texttt{f} or \texttt{g}

The input substring must be a floating-point number consisting of an optional sign, a string of decimal digits possibly containing a decimal point, and an optional exponent consisting of an \texttt{e} or \texttt{E} followed by an optional sign and a string of decimal digits. It is read in and stored in the variable as a floating-point value.

[\texttt{chars}]

The input substring consists of one or more characters in \texttt{chars}. The matching string is stored in the variable. If the first character between the brackets is a \texttt{]} then it is treated as part of \texttt{chars} rather than the closing bracket for the set. If \texttt{chars} contains a sequence of the form \texttt{a-b} then any character between \texttt{a} and \texttt{b} (inclusive) will match. If the first or last character between the brackets is a \texttt{-}, then it is treated as part of \texttt{chars} rather than indicating a range.
The input substring consists of one or more characters not in \textit{chars}. The matching string is stored in the variable. If the character immediately following the ^ is a ] then it is treated as part of the set rather than the closing bracket for the set. If \textit{chars} contains a sequence of the form \texttt{a-b} then any character between \texttt{a} and \texttt{b} (inclusive) will be excluded from the set. If the first or last character between the brackets is a - , then it is treated as part of \textit{chars} rather than indicating a range value.

\textbf{n}

No input is consumed from the input string. Instead, the total number of characters scanned from the input string so far is stored in the variable.

The number of characters read from the input for a conversion is the largest number that makes sense for that particular conversion (e.g. as many decimal digits as possible for \texttt{\%d}, as many octal digits as possible for \texttt{\%o}, and so on). The input substring for a given conversion terminates either when a white-space character is encountered or when the maximum substring width has been reached, whichever comes first. If a * is present in the conversion specifier then no variable is assigned and the next scan argument is not consumed.

\textbf{DIFFERENCES FROM ANSI SSCANF}

The behavior of the \texttt{scan} command is the same as the behavior of the ANSI C \texttt{sscanf} procedure except for the following differences:

\textbf{[1]}

\texttt{\%p} conversion specifier is not supported.

\textbf{[2]}

For \texttt{\%c} conversions a single character value is converted to a decimal string, which is then assigned to the corresponding \texttt{varName}; no substring width may be specified for this conversion.

\textbf{[3]}
The h modifier is always ignored and the l and L modifiers are ignored when converting real values (i.e. type double is used for the internal representation). The ll modifier has no scanf counterpart.

[4]

If the end of the input string is reached before any conversions have been performed and no variables are given, an empty string is returned.

EXAMPLES

Convert a UNICODE character to its numeric value:

```
set char "x"
set value [scan $char %c]
```

Parse a simple color specification of the form #RRGGBB using hexadecimal conversions with substring sizes:

```
set string "#08D03F"
scan $string "#%2x%2x%2x" r g b
```

Parse a HH:MM time string, noting that this avoids problems with octal numbers by forcing interpretation as decimals (if we did not care, we would use the %i conversion instead):

```
set string "08:08" ;# *Not* octal!
if {[scan $string "%d:%d" hours minutes] != 2} {
    error "not a valid time string"
}
# We have to understand numeric ranges ourselves...
if {($minutes < 0 || $minutes > 59) { 
    error "invalid number of minutes"
```
Break a string up into sequences of non-whitespace characters (note the use of the \%n conversion so that we get skipping over leading whitespace correct):

```tcl
set string "a string {with braced words} + leading
set words {}
while {[\texttt{scan} $string \%s\%n word length] == 2} {
  lappend words $word
  set string [string range $string $length end]
}
```

Parse a simple coordinate string, checking that it is complete by looking for the terminating character explicitly:

```tcl
set string "(5.2,-4e-2)"
# Note that the spaces before the literal parts of
# the scan pattern are significant, and that ")" is
# the Unicode character \u0029
if {
  [\texttt{scan} $string " (%f ,%f %c" x y last] != 3
  || $last != 0x0029
} then {
  error "invalid coordinate string"
}
puts "X=$x, Y=$y"
```

An interactive session demonstrating the truncation of integer values determined by size modifiers:
% set tcl_platform(wordSize)
4
% scan 20000000000000000000 %d
2147483647
% scan 20000000000000000000 %ld
9223372036854775807
% scan 20000000000000000000 %lld
20000000000000000000

SEE ALSO

format, scanf

KEYWORDS

conversion specifier, parse, scan

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 2000 Scriptics Corporation.
NAME
update - Process pending events and idle callbacks

SYNOPSIS
update ?idletasks?

DESCRIPTION
This command is used to bring the application “up to date” by entering the event loop repeatedly until all pending events (including idle callbacks) have been processed.

If the idletasks keyword is specified as an argument to the command, then no new events or errors are processed; only idle callbacks are invoked. This causes operations that are normally deferred, such as display updates and window layout calculations, to be performed immediately.

The update idletasks command is useful in scripts where changes have been made to the application's state and you want those changes to appear on the display immediately, rather than waiting for the script to complete. Most display updates are performed as idle callbacks, so update idletasks will cause them to run. However, there are some kinds of updates that only happen in response to events, such as those triggered by window size changes; these updates will not occur in update idletasks.

The update command with no options is useful in scripts where you are performing a long-running computation but you still want the application to respond to events such as user interactions; if you occasionally call
The `update` command then user input will be processed during the next call to `update`.

**EXAMPLE**

Run computations for about a second and then finish:

```bash
set x 1000
set done 0
after 1000 set done 1
while {!$done} {
    # A very silly example!
    set x [expr {log($x) ** 2.8}]
    # Test to see if our time-limit has been hit. T
    # also give a chance for serving network sockets
    # the Tk package is loaded, updating a user inte
    update
}
```

**SEE ALSO**

`after`, `interp`

**KEYWORDS**

`event`, `flush`, `handler`, `idle`, `update`
NAME

cd - Change working directory

SYNOPSIS

```
cd ?dirName?
```

DESCRIPTION

Change the current working directory to `dirName`, or to the home directory (as specified in the HOME environment variable) if `dirName` is not given. Returns an empty string. Note that the current working directory is a per-process resource; the `cd` command changes the working directory for all interpreters and (in a threaded environment) all threads.

EXAMPLES

Change to the home directory of the user `fred`:

```
cd ~fred
```

Change to the directory `lib` that is a sibling directory of the current one:

```
cd ../lib
```
filename, glob, pwd

KEYWORDS

working directory

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

gets - Read a line from a channel

SYNOPSIS

gets channelId ?varName?

DESCRIPTION

This command reads the next line from channelId, returns everything in the line up to (but not including) the end-of-line character(s), and discards the end-of-line character(s).

ChannelId must be an identifier for an open channel such as the Tcl standard input channel (stdin), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension. The channel must have been opened for input.

If varName is omitted the line is returned as the result of the command. If varName is specified then the line is placed in the variable by that name and the return value is a count of the number of characters returned.

If end of file occurs while scanning for an end of line, the command returns whatever input is available up to the end of file. If channelId is in nonblocking mode and there is not a full line of input available, the command returns an empty string and does not consume any input. If varName is specified and an empty string is returned in varName because of end-of-file or because of insufficient data in nonblocking mode, then the return count is -1. Note that if varName is not specified then the end-of-file and no-full-line-available cases can produce the
same results as if there were an input line consisting only of the end-of-line character(s). The `eof` and `fblocked` commands can be used to distinguish these three cases.

**EXAMPLE**

This example reads a file one line at a time and prints it out with the current line number attached to the start of each line.

```tcl
set chan [open "some.file.txt"]
set lineNumber 0
while {[gets $chan line] >= 0} {
    puts "[incr lineNumber]: $line"
}
close $chan
```

**SEE ALSO**

`file`, `eof`, `fblocked`, `Tcl_StandardChannels`

**KEYWORDS**

blocking, channel, end of file, end of line, line, nonblocking, read

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
mathop - Mathematical operators as Tcl commands

SYNOPSIS

DESCRIPTION

MATHEMATICAL OPERATORS

! boolean
+ ?number ...?
- number ?number ...?
* ?number ...
/ ?number ...
% ?number ?number
** ?number ...

COMPARISON OPERATORS

== ?arg ...
eq ?arg ...
!= ?arg ?arg
eq ?arg ?arg
< ?arg ...
<= ?arg ...
> ?arg ...
>= ?arg ...

BIT-WISE OPERATORS

~ ?number
& ?number ...
| ?number ...
<< ?number ?number
>> ?number ?number

LIST OPERATORS

in ?arg ?list
ni ?arg ?list

EXAMPLES
NAME

mathop - Mathematical operators as Tcl commands

SYNOPSIS

package require Tcl 8.5
::tcl::mathop::! number
::tcl::mathop::~ number
::tcl::mathop::+ ?number ...
::tcl::mathop::- number ?number ...
::tcl::mathop::* ?number ...
::tcl::mathop::/ number ?number ...
::tcl::mathop::% number number
::tcl::mathop::** ?number ...
::tcl::mathop::& ?number ...
::tcl::mathop::| ?number ...
::tcl::mathop::^ ?number ...
::tcl::mathop::* number number
::tcl::mathop::>= ?arg ...
::tcl::mathop::== ?arg ...
::tcl::mathop::!= arg arg
::tcl::mathop::< ?arg ...
::tcl::mathop::<= ?arg ...
::tcl::mathop::>= ?arg ...
::tcl::mathop::> ?arg ...
::tcl::mathop::eq ?arg ...
::tcl::mathop::ne arg arg
::tcl::mathop::in arg list
::tcl::mathop::ni arg list

DESCRIPTION

The commands in the ::tcl::mathop namespace implement the same set of operations as supported by the expr command. All are exported
from the namespace, but are not imported into any other namespace by default. Note that renaming, reimplementing or deleting any of the commands in the namespace does not alter the way that the `expr` command behaves, and nor does defining any new commands in the `::tcl::mathop` namespace.

The following operator commands are supported:

```
~   !   +   -   *

/   %   **   &   |

^   >>   <<   ==  eq

!=  ne <  <=  >

>=  in  ni
```

### MATHEMATICAL OPERATORS

The behaviors of the mathematical operator commands are as follows:

**! boolean**

Returns the boolean negation of `boolean`, where `boolean` may be any numeric value or any other form of boolean value (i.e. it returns truth if the argument is falsity or zero, and falsity if the argument is truth or non-zero).

**+ ?number ...?**

Returns the sum of arbitrarily many arguments. Each `number` argument may be any numeric value. If no arguments are given, the result will be zero (the summation identity).
- `number ?number ...?`  
  If only a single `number` argument is given, returns the negation of that numeric value. Otherwise returns the number that results when all subsequent numeric values are subtracted from the first one. All `number` arguments must be numeric values. At least one argument must be given.

* `?number ...?`  
  Returns the product of arbitrarily many arguments. Each `number` may be any numeric value. If no arguments are given, the result will be one (the multiplicative identity).

I `number ?number ...?`  
If only a single `number` argument is given, returns the reciprocal of that numeric value (i.e. the value obtained by dividing 1.0 by that value). Otherwise returns the number that results when the first numeric argument is divided by all subsequent numeric arguments. All `number` arguments must be numeric values. At least one argument must be given.

Note that when the leading values in the list of arguments are integers, integer division will be used for those initial steps (i.e. the intermediate results will be as if the functions `floor` and `int` are applied to them, in that order). If all values in the operation are integers, the result will be an integer.

% `number number`  
Returns the integral modulus of the first argument with respect to the second. Each `number` must have an integral value. Note that Tcl defines this operation exactly even for negative numbers, so that the following equality holds true:

\[(x / y) \times y = x - (x \% y)\]

** `?number ...?`  
Returns the result of raising each value to the power of the result of
recursively operating on the result of processing the following arguments, so "** 2 3 4" is the same as "** 2 [** 3 4]". Each number may be any numeric value, though the second number must not be fractional if the first is negative. If no arguments are given, the result will be one, and if only one argument is given, the result will be that argument. The result will have an integral value only when all arguments are integral values.

**COMPARISON OPERATORS**

The behaviors of the comparison operator commands (most of which operate preferentially on numeric arguments) are as follows:

**== ?arg ...?**
Returns whether each argument is equal to the arguments on each side of it in the sense of the expr == operator (i.e., numeric comparison if possible, exact string comparison otherwise). If fewer than two arguments are given, this operation always returns a true value.

**eq ?arg ...?**
Returns whether each argument is equal to the arguments on each side of it using exact string comparison. If fewer than two arguments are given, this operation always returns a true value.

**!= arg arg**
Returns whether the two arguments are not equal to each other, in the sense of the expr != operator (i.e., numeric comparison if possible, exact string comparison otherwise).

**ne arg arg**
Returns whether the two arguments are not equal to each other using exact string comparison.

**< ?arg ...?**
Returns whether the arbitrarily-many arguments are ordered, with each argument after the first having to be strictly more than the one preceding it. Comparisons are performed preferentially on the
numeric values, and are otherwise performed using UNICODE string comparison. If fewer than two arguments are present, this operation always returns a true value. When the arguments are numeric but should be compared as strings, the string compare command should be used instead.

\[ \leq \text{arg ...?} \]
Returns whether the arbitrarily-many arguments are ordered, with each argument after the first having to be equal to or more than the one preceding it. Comparisons are performed preferentially on the numeric values, and are otherwise performed using UNICODE string comparison. If fewer than two arguments are present, this operation always returns a true value. When the arguments are numeric but should be compared as strings, the string compare command should be used instead.

\[ > \text{arg ...?} \]
Returns whether the arbitrarily-many arguments are ordered, with each argument after the first having to be strictly less than the one preceding it. Comparisons are performed preferentially on the numeric values, and are otherwise performed using UNICODE string comparison. If fewer than two arguments are present, this operation always returns a true value. When the arguments are numeric but should be compared as strings, the string compare command should be used instead.

\[ \geq \text{arg ...?} \]
Returns whether the arbitrarily-many arguments are ordered, with each argument after the first having to be equal to or less than the one preceding it. Comparisons are performed preferentially on the numeric values, and are otherwise performed using UNICODE string comparison. If fewer than two arguments are present, this operation always returns a true value. When the arguments are numeric but should be compared as strings, the string compare command should be used instead.
The behaviors of the bit-wise operator commands (all of which only operate on integral arguments) are as follows:

\[ \sim \text{number} \]
Returns the bit-wise negation of \text{number}. \text{Number} may be an integer of any size. Note that the result of this operation will always have the opposite sign to the input \text{number}.

\& \ ?\text{number} ...? 
Returns the bit-wise AND of each of the arbitrarily many arguments. Each \text{number} must have an integral value. If no arguments are given, the result will be minus one.

\mid \ ?\text{number} ...? 
Returns the bit-wise OR of each of the arbitrarily many arguments. Each \text{number} must have an integral value. If no arguments are given, the result will be zero.

\ll\text{number} \text{number} 
Returns the result of bit-wise shifting the first argument left by the number of bits specified in the second argument. Each \text{number} must have an integral value.

\gg\text{number} \text{number} 
Returns the result of bit-wise shifting the first argument right by the number of bits specified in the second argument. Each \text{number} must have an integral value.

\text{LIST OPERATORS}

The behaviors of the list-oriented operator commands are as follows:

\text{in} \ \text{arg} \ \text{list} 
Returns whether the value \text{arg} is present in the list \text{list} (according to exact string comparison of elements).
ni arg list
Returns whether the value arg is not present in the list list (according to exact string comparison of elements).

**EXAMPLES**

The simplest way to use the operators is often by using namespace path to make the commands available. This has the advantage of not affecting the set of commands defined by the current namespace.

```tcl	namespace path {::tcl::mathop ::tcl::mathfunc}

# Compute the sum of some numbers
set sum [+ 1 2 3]

# Compute the average of a list
set list {1 2 3 4 5 6}
set mean [/ [+ {*}$list] [double [llength $list]]]

# Test for list membership
set gotIt [in 3 $list]

# Test to see if a value is within some defined range
set inRange [<= 1 $x 5]

# Test to see if a list is sorted
set sorted [<= {*}$list]
```

**SEE ALSO**

expr, mathfunc, namespace

**KEYWORDS**

command, expression, operator
NAME

seek - Change the access position for an open channel

SYNOPSIS

seek channelId offset ?origin?

DESCRIPTION

Changes the current access position for channelId.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

The offset and origin arguments specify the position at which the next read or write will occur for channelId. Offset must be an integer (which may be negative) and origin must be one of the following:

start
   The new access position will be offset bytes from the start of the underlying file or device.

current
   The new access position will be offset bytes from the current access position; a negative offset moves the access position backwards in the underlying file or device.

end
   The new access position will be offset bytes from the end of the file or device. A negative offset places the access position before the
end of file, and a positive \textit{offset} places the access position after the end of file.

The \textit{origin} argument defaults to \textbf{start}.

The command flushes all buffered output for the channel before the command returns, even if the channel is in nonblocking mode. It also discards any buffered and unread input. This command returns an empty string. An error occurs if this command is applied to channels whose underlying file or device does not support seeking.

Note that \textit{offset} values are byte offsets, not character offsets. Both \texttt{seek} and \texttt{tell} operate in terms of bytes, not characters, unlike \texttt{read}.

\begin{table}[h]
\centering
\begin{tabular}{ll}
\hline
\textbf{EXAMPLES} & \\
\hline
Read a file twice: & \\
\hline
\begin{verbatim}
set f [open file.txt]
set data1 [read $f]
\textbf{seek} $f 0
set data2 [read $f]
close $f
# $data1 == $data2 if the file wasn't updated
\end{verbatim} & \\
\hline
Read the last 10 bytes from a file: & \\
\hline
\begin{verbatim}
set f [open file.data]
# This is guaranteed to work with binary data but
# may fail with other encodings...
fconfigure $f -translation binary
\textbf{seek} $f -10 end
set data [read $f 10]
close $f
\end{verbatim} & \\
\hline
\end{tabular}
\caption{Examples of using the \texttt{seek} command.}
\end{table}
SEE ALSO

file, open, close, gets, tell, Tcl_StandardChannels

KEYWORDS

access position, file, seek

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

uplevel - Execute a script in a different stack frame

SYNOPSIS

uplevel ?level? arg ?arg ...?

DESCRIPTION

All of the arg arguments are concatenated as if they had been passed to concat; the result is then evaluated in the variable context indicated by level. Uplevel returns the result of that evaluation.

If level is an integer then it gives a distance (up the procedure calling stack) to move before executing the command. If level consists of # followed by a number then the number gives an absolute level number. If level is omitted then it defaults to 1. Level cannot be defaulted if the first command argument starts with a digit or #.

For example, suppose that procedure a was invoked from top-level, and that it called b, and that b called c. Suppose that c invokes the uplevel command. If level is 1 or #2 or omitted, then the command will be executed in the variable context of b. If level is 2 or #1 then the command will be executed in the variable context of a. If level is 3 or #0 then the command will be executed at top-level (only global variables will be visible).

The uplevel command causes the invoking procedure to disappear from the procedure calling stack while the command is being executed. In the above example, suppose c invokes the command
where \texttt{d} is another Tcl procedure. The \texttt{set} command will modify the
variable \texttt{x} in \texttt{b}'s context, and \texttt{d} will execute at level 3, as if called from
\texttt{b}. If it in turn executes the command

\begin{verbatim}
uplevel 1 \{set x 43; d\}
\end{verbatim}

then the \texttt{set} command will modify the same variable \texttt{x} in \texttt{b}'s context:
the procedure \texttt{c} does not appear to be on the call stack when \texttt{d} is
executing. The \texttt{info level} command may be used to obtain the level of
the current procedure.

\textbf{Uplevel} makes it possible to implement new control constructs as Tcl
procedures (for example, \texttt{uplevel} could be used to implement the \texttt{while}
construct as a Tcl procedure).

The \texttt{namespace eval} and \texttt{apply} commands offer other ways (besides
procedure calls) that the Tcl naming context can change. They add a
call frame to the stack to represent the namespace context. This means
each \texttt{namespace eval} command counts as another call level for
\texttt{uplevel} and \texttt{upvar} commands. For example, \texttt{info level 1} will return a
list describing a command that is either the outermost procedure call or
the outermost \texttt{namespace eval} command. Also, \texttt{uplevel \#0} evaluates
a script at top-level in the outermost namespace (the global
namespace).

\textbf{EXAMPLE}

As stated above, the \texttt{uplevel} command is useful for creating new
control constructs. This example shows how (without error handling) it
can be used to create a \texttt{do} command that is the counterpart of \texttt{while}
except for always performing the test after running the loop body:

\begin{verbatim}
proc do \{body while condition\} {
\end{verbatim}
if {$while ne "while"} {
    error "required word missing"
}
set conditionCmd [list expr $condition]
while {1} {
    uplevel 1 $body
    if {![uplevel 1 $conditionCmd]} {
        break
    }
}

SEE ALSO

apply, namespace, upvar

KEYWORDS

class, level, namespace, stack frame, variables

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME
chan - Read, write and manipulate channels

SYNOPSIS

DESCRIPTION
chan blocked channelId
chan close channelId
chan configure channelId ?optionName? ?value? ?
    optionName value?...
    -blocking boolean
    -buffering newValue
    -buffersize newSize
    -encoding name
    -eofchar char
    -eofchar {inChar outChar}
    -translation mode
    -translation {inMode outMode}
        auto
        binary
        cr
        crlf
    If
chan copy inputChan outputChan ?-size size? ?-command callback?
chan create mode cmdPrefix
chan eof channelId
chan event channelId event ?script?
chan flush channelId
chan gets channelId ?varName?
chan names ?pattern?
chan pending mode channelId
chan postevent channelId eventSpec
**NAME**

`chan` - Read, write and manipulate channels

**SYNOPSIS**

```
chan option ?arg arg ...?
```

**DESCRIPTION**

This command provides several operations for reading from, writing to and otherwise manipulating open channels (such as have been created with the `open` and `socket` commands, or the default named channels `stdin`, `stdout` or `stderr` which correspond to the process’s standard input, output and error streams respectively). `Option` indicates what to do with the channel; any unique abbreviation for `option` is acceptable. Valid options are:

**chan blocked channelId**

This tests whether the last input operation on the channel called `channelId` failed because it would have otherwise caused the process to block, and returns 1 if that was the case. It returns 0 otherwise. Note that this only ever returns 1 when the channel has
been configured to be non-blocking; all Tcl channels have blocking turned on by default.

**chan close channelld**

Close and destroy the channel called *channelld*. Note that this deletes all existing file-events registered on the channel.

As part of closing the channel, all buffered output is flushed to the channel's output device, any buffered input is discarded, the underlying operating system resource is closed and *channelld* becomes unavailable for future use.

If the channel is blocking, the command does not return until all output is flushed. If the channel is nonblocking and there is unflushed output, the channel remains open and the command returns immediately; output will be flushed in the background and the channel will be closed when all the flushing is complete.

If *channelld* is a blocking channel for a command pipeline then **chan close** waits for the child processes to complete.

If the channel is shared between interpreters, then **chan close** makes *channelld* unavailable in the invoking interpreter but has no other effect until all of the sharing interpreters have closed the channel. When the last interpreter in which the channel is registered invokes **chan close** (or **close**), the cleanup actions described above occur. See the **interp** command for a description of channel sharing.

Channels are automatically closed when an interpreter is destroyed and when the process exits. Channels are switched to blocking mode, to ensure that all output is correctly flushed before the process exits.

The command returns an empty string, and may generate an error if an error occurs while flushing output. If a command in a command pipeline created with **open** returns an error, **chan close** generates an error (similar to the **exec** command.)
**chan configure** `channelId ?optionName? ?value? ?optionName value?...

Query or set the configuration options of the channel named `channelId`.

If no `optionName` or `value` arguments are supplied, the command returns a list containing alternating option names and values for the channel. If `optionName` is supplied but no `value` then the command returns the current value of the given option. If one or more pairs of `optionName` and `value` are supplied, the command sets each of the named options to the corresponding `value`; in this case the return value is an empty string.

The options described below are supported for all channels. In addition, each channel type may add options that only it supports. See the manual entry for the command that creates each type of channels for the options that that specific type of channel supports. For example, see the manual entry for the `socket` command for its additional options.

**-blocking** boolean

The **-blocking** option determines whether I/O operations on the channel can cause the process to block indefinitely. The value of the option must be a proper boolean value. Channels are normally in blocking mode; if a channel is placed into nonblocking mode it will affect the operation of the `chan gets`, `chan read`, `chan puts`, `chan flush`, and `chan close` commands; see the documentation for those commands for details. For nonblocking mode to work correctly, the application must be using the Tcl event loop (e.g. by calling `Tcl_DoOneEvent` or invoking the `vwait` command).

**-buffering** `newValue`

If `newValue` is **full** then the I/O system will buffer output until its internal buffer is full or until the `chan flush` command is invoked. If `newValue` is **line**, then the I/O system will automatically flush output for the channel whenever a newline character is output. If `newValue` is **none**, the I/O system will
flush automatically after every output operation. The default is for `-buffering` to be set to `full` except for channels that connect to terminal-like devices; for these channels the initial setting is `line`. Additionally, `stdin` and `stdout` are initially set to `line`, and `stderr` is set to `none`.

- `buffersize newSize`
  `Newvalue` must be an integer; its value is used to set the size of buffers, in bytes, subsequently allocated for this channel to store input or output. `Newvalue` must be a number of no more than one million, allowing buffers of up to one million bytes in size.

- `encoding name`
  This option is used to specify the encoding of the channel as one of the named encodings returned by `encoding names` or the special value `binary`, so that the data can be converted to and from Unicode for use in Tcl. For instance, in order for Tcl to read characters from a Japanese file in `shiftjis` and properly process and display the contents, the encoding would be set to `shiftjis`. Thereafter, when reading from the channel, the bytes in the Japanese file would be converted to Unicode as they are read. Writing is also supported - as Tcl strings are written to the channel they will automatically be converted to the specified encoding on output.

If a file contains pure binary data (for instance, a JPEG image), the encoding for the channel should be configured to be `binary`. Tcl will then assign no interpretation to the data in the file and simply read or write raw bytes. The Tcl `binary` command can be used to manipulate this byte-oriented data. It is usually better to set the `-translation` option to `binary` when you want to transfer binary data, as this turns off the other automatic interpretations of the bytes in the stream as well.

The default encoding for newly opened channels is the same platform- and locale-dependent system encoding used for interfacing with the operating system, as returned by `encoding`
-eofchar char

-eofchar {inChar outChar}
This option supports DOS file systems that use Control-z (\x1a) as an end of file marker. If char is not an empty string, then this character signals end-of-file when it is encountered during input. For output, the end-of-file character is output when the channel is closed. If char is the empty string, then there is no special end of file character marker. For read-write channels, a two-element list specifies the end of file marker for input and output, respectively. As a convenience, when setting the end-of-file character for a read-write channel you can specify a single value that will apply to both reading and writing. When querying the end-of-file character of a read-write channel, a two-element list will always be returned. The default value for -eofchar is the empty string in all cases except for files under Windows. In that case the -eofchar is Control-z (\x1a) for reading and the empty string for writing. The acceptable range for -eofchar values is \x01 - \x7f; attempting to set -eofchar to a value outside of this range will generate an error.

-translation mode

-translation {inMode outMode}
In Tcl scripts the end of a line is always represented using a single newline character (\n). However, in actual files and devices the end of a line may be represented differently on different platforms, or even for different devices on the same platform. For example, under UNIX newlines are used in files, whereas carriage-return-linefeed sequences are normally used in network connections. On input (i.e., with chan gets and chan read) the Tcl I/O system automatically translates the external end-of-line representation into newline characters. Upon output (i.e., with chan puts), the I/O system translates newlines to the external end-of-line representation. The default
translation mode, **auto**, handles all the common cases automatically, but the **-translation** option provides explicit control over the end of line translations.

The value associated with **-translation** is a single item for read-only and write-only channels. The value is a two-element list for read-write channels; the read translation mode is the first element of the list, and the write translation mode is the second element. As a convenience, when setting the translation mode for a read-write channel you can specify a single value that will apply to both reading and writing. When querying the translation mode of a read-write channel, a two-element list will always be returned. The following values are currently supported:

**auto**

As the input translation mode, **auto** treats any of newline (lf), carriage return (cr), or carriage return followed by a newline (crlf) as the end of line representation. The end of line representation can even change from line-to-line, and all cases are translated to a newline. As the output translation mode, **auto** chooses a platform specific representation; for sockets on all platforms Tcl chooses crlf, for all Unix flavors, it chooses lf, and for the various flavors of Windows it chooses crlf. The default setting for **-translation** is **auto** for both input and output.

**binary**

No end-of-line translations are performed. This is nearly identical to **lf** mode, except that in addition **binary** mode also sets the end-of-file character to the empty string (which disables it) and sets the encoding to **binary** (which disables encoding filtering). See the description of **-eofchar** and **-encoding** for more information.

**cr**

The end of a line in the underlying file or device is represented by a single carriage return character. As the
input translation mode, **cr** mode converts carriage returns to newline characters. As the output translation mode, **cr** mode translates newline characters to carriage returns.

**crlf**

The end of a line in the underlying file or device is represented by a carriage return character followed by a linefeed character. As the input translation mode, **crlf** mode converts carriage-return-linefeed sequences to newline characters. As the output translation mode, **crlf** mode translates newline characters to carriage-return-linefeed sequences. This mode is typically used on Windows platforms and for network connections.

**lf**

The end of a line in the underlying file or device is represented by a single newline (linefeed) character. In this mode no translations occur during either input or output. This mode is typically used on UNIX platforms.

**chan copy** *inputChan* *outputChan* ?-*size* *size*? ?-*command* *callback*?

Copy data from the channel *inputChan*, which must have been opened for reading, to the channel *outputChan*, which must have been opened for writing. The **chan copy** command leverages the buffering in the Tcl I/O system to avoid extra copies and to avoid buffering too much data in main memory when copying large files to slow destinations like network sockets.

The **chan copy** command transfers data from *inputChan* until end of file or size bytes have been transferred. If no -size argument is given, then the copy goes until end of file. All the data read from *inputChan* is copied to *outputChan*. Without the -command option, **chan copy** blocks until the copy is complete and returns the number of bytes written to *outputChan*.

The -command argument makes **chan copy** work in the background. In this case it returns immediately and the *callback* is invoked later when the copy completes. The *callback* is called with
one or two additional arguments that indicates how many bytes were written to `outputChan`. If an error occurred during the background copy, the second argument is the error string associated with the error. With a background copy, it is not necessary to put `inputChan` or `outputChan` into non-blocking mode; the `chan copy` command takes care of that automatically. However, it is necessary to enter the event loop by using the `vwait` command or by using Tk.

You are not allowed to do other I/O operations with `inputChan` or `outputChan` during a background `chan copy`. If either `inputChan` or `outputChan` get closed while the copy is in progress, the current copy is stopped and the command callback is not made. If `inputChan` is closed, then all data already queued for `outputChan` is written out.

Note that `inputChan` can become readable during a background copy. You should turn off any `chan event` or `fileevent` handlers during a background copy so those handlers do not interfere with the copy. Any I/O attempted by a `chan event` or `fileevent` handler will get a “channel busy” error.

`Chan copy` translates end-of-line sequences in `inputChan` and `outputChan` according to the `-translation` option for these channels (see `chan configure` above). The translations mean that the number of bytes read from `inputChan` can be different than the number of bytes written to `outputChan`. Only the number of bytes written to `outputChan` is reported, either as the return value of a synchronous `chan copy` or as the argument to the callback for an asynchronous `chan copy`.

`Chan copy` obeys the encodings and character translations configured for the channels. This means that the incoming characters are converted internally first UTF-8 and then into the encoding of the channel `chan copy` writes to (see `chan configure` above for details on the `-encoding` and `-translation` options). No conversion is done if both channels are set to encoding `binary` and have matching translations. If only the output channel is set to
encoding **binary** the system will write the internal UTF-8 representation of the incoming characters. If only the input channel is set to encoding **binary** the system will assume that the incoming bytes are valid UTF-8 characters and convert them according to the output encoding. The behaviour of the system for bytes which are not valid UTF-8 characters is undefined in this case.

**chan create mode cmdPrefix**

This subcommand creates a new script level channel using the command prefix *cmdPrefix* as its handler. Any such channel is called a **reflected** channel. The specified command prefix, *cmdPrefix*, must be a non-empty list, and should provide the API described in the **reflectedchan** manual page. The handle of the new channel is returned as the result of the **chan create** command, and the channel is open. Use either **close** or **chan close** to remove the channel.

The argument *mode* specifies if the new channel is opened for reading, writing, or both. It has to be a list containing any of the strings “**read**” or “**write**”. The list must have at least one element, as a channel you can neither write to nor read from makes no sense. The handler command for the new channel must support the chosen mode, or an error is thrown.

The command prefix is executed in the global namespace, at the top of call stack, following the appending of arguments as described in the **reflectedchan** manual page. Command resolution happens at the time of the call. Renaming the command, or destroying it means that the next call of a handler method may fail, causing the channel command invoking the handler to fail as well. Depending on the subcommand being invoked, the error message may not be able to explain the reason for that failure.

Every channel created with this subcommand knows which interpreter it was created in, and only ever executes its handler command in that interpreter, even if the channel was shared with and/or was moved into a different interpreter. Each reflected channel also knows the thread it was created in, and executes its
handler command only in that thread, even if the channel was moved into a different thread. To this end all invocations of the handler are forwarded to the original thread by posting special events to it. This means that the original thread (i.e. the thread that executed the `chan create` command) must have an active event loop, i.e. it must be able to process such events. Otherwise the thread sending them will block indefinitely. Deadlock may occur.

Note that this permits the creation of a channel whose two endpoints live in two different threads, providing a stream-oriented bridge between these threads. In other words, we can provide a way for regular stream communication between threads instead of having to send commands.

When a thread or interpreter is deleted, all channels created with this subcommand and using this thread/interpreter as their computing base are deleted as well, in all interpreters they have been shared with or moved into, and in whatever thread they have been transferred to. While this pulls the rug out under the other thread(s) and/or interpreter(s), this cannot be avoided. Trying to use such a channel will cause the generation of a regular error about unknown channel handles.

This subcommand is **safe** and made accessible to safe interpreters. While it arranges for the execution of arbitrary Tcl code the system also makes sure that the code is always executed within the safe interpreter.

**chan eof channelId**

Test whether the last input operation on the channel called `channelId` failed because the end of the data stream was reached, returning 1 if end-of-file was reached, and 0 otherwise.

**chan event channelId event script?**

 Arrange for the Tcl script `script` to be installed as a *file event handler* to be called whenever the channel called `channelId` enters the state described by `event` (which must be either `readable` or `writable`); only one such handler may be installed per event per
channel at a time. If *script* is the empty string, the current handler is deleted (this also happens if the channel is closed or the interpreter deleted). If *script* is omitted, the currently installed script is returned (or an empty string if no such handler is installed). The callback is only performed if the event loop is being serviced (e.g. via *vwait* or *update*).

A file event handler is a binding between a channel and a script, such that the script is evaluated whenever the channel becomes readable or writable. File event handlers are most commonly used to allow data to be received from another process on an event-driven basis, so that the receiver can continue to interact with the user or with other channels while waiting for the data to arrive. If an application invokes *chan gets* or *chan read* on a blocking channel when there is no input data available, the process will block; until the input data arrives, it will not be able to service other events, so it will appear to the user to “freeze up”. With *chan event*, the process can tell when data is present and only invoke *chan gets* or *chan read* when they will not block.

A channel is considered to be readable if there is unread data available on the underlying device. A channel is also considered to be readable if there is unread data in an input buffer, except in the special case where the most recent attempt to read from the channel was a *chan gets* call that could not find a complete line in the input buffer. This feature allows a file to be read a line at a time in nonblocking mode using events. A channel is also considered to be readable if an end of file or error condition is present on the underlying file or device. It is important for *script* to check for these conditions and handle them appropriately; for example, if there is no special check for end of file, an infinite loop may occur where *script* reads no data, returns, and is immediately invoked again.

A channel is considered to be writable if at least one byte of data can be written to the underlying file or device without blocking, or if an error condition is present on the underlying file or device. Note that client sockets opened in asynchronous mode become writable when they become connected or if the connection fails.
Event-driven I/O works best for channels that have been placed into nonblocking mode with the `chan configure` command. In blocking mode, a `chan puts` command may block if you give it more data than the underlying file or device can accept, and a `chan gets` or `chan read` command will block if you attempt to read more data than is ready; no events will be processed while the commands block. In nonblocking mode `chan puts`, `chan read`, and `chan gets` never block.

The script for a file event is executed at global level (outside the context of any Tcl procedure) in the interpreter in which the `chan event` command was invoked. If an error occurs while executing the script then the command registered with `interp bgerror` is used to report the error. In addition, the file event handler is deleted if it ever returns an error; this is done in order to prevent infinite loops due to buggy handlers.

`chan flush channelId`
Ensures that all pending output for the channel called `channelId` is written.

If the channel is in blocking mode the command does not return until all the buffered output has been flushed to the channel. If the channel is in nonblocking mode, the command may return before all buffered output has been flushed; the remainder will be flushed in the background as fast as the underlying file or device is able to absorb it.

`chan gets channelId ?varName?`
Reads the next line from the channel called `channelId`. If `varName` is not specified, the result of the command will be the line that has been read (without a trailing newline character) or an empty string upon end-of-file or, in non-blocking mode, if the data available is exhausted. If `varName` is specified, the line that has been read will be written to the variable called `varName` and result will be the number of characters that have been read or -1 if end-of-file was reached or, in non-blocking mode, if the data available is
exhausted.

If an end-of-file occurs while part way through reading a line, the partial line will be returned (or written into \texttt{varName}). When \texttt{varName} is not specified, the end-of-file case can be distinguished from an empty line using the \texttt{chan eof} command, and the partial-line-but-nonblocking case can be distinguished with the \texttt{chan blocked} command.

\textbf{chan names} \texttt{?pattern}\texttt{?}

Produces a list of all channel names. If \texttt{pattern} is specified, only those channel names that match it (according to the rules of \texttt{string match}) will be returned.

\textbf{chan pending} \texttt{mode channelId}

Depending on whether \texttt{mode} is \texttt{input} or \texttt{output}, returns the number of bytes of input or output (respectively) currently buffered internally for \texttt{channelId} (especially useful in a readable event callback to impose application-specific limits on input line lengths to avoid a potential denial-of-service attack where a hostile user crafts an extremely long line that exceeds the available memory to buffer it). Returns -1 if the channel was not opened for the mode in question.

\textbf{chan postevent} \texttt{channelId eventSpec}

This subcommand is used by command handlers specified with \texttt{chan create}. It notifies the channel represented by the handle \texttt{channelId} that the event(s) listed in the \texttt{eventSpec} have occurred. The argument has to be a list containing any of the strings \texttt{read} and \texttt{write}. The list must contain at least one element as it does not make sense to invoke the command if there are no events to post.

Note that this subcommand can only be used with channel handles that were created/opened by \texttt{chan create}. All other channels will cause this subcommand to report an error.

As only the Tcl level of a channel, i.e. its command handler, should post events to it we also restrict the usage of this command to the
interpreter that created the channel. In other words, posting events to a reflected channel from an interpreter that does not contain it's implementation is not allowed. Attempting to post an event from any other interpreter will cause this subcommand to report an error.

Another restriction is that it is not possible to post events that the I/O core has not registered an interest in. Trying to do so will cause the method to throw an error. See the command handler method watch described in reflectedchan, the document specifying the API of command handlers for reflected channels.

This command is safe and made accessible to safe interpreters. It can trigger the execution of chan event handlers, whether in the current interpreter or in other interpreters or other threads, even where the event is posted from a safe interpreter and listened for by a trusted interpreter. Chan event handlers are always executed in the interpreter that set them up.

chan puts ?-nonewline? ?channelId? string

 Writes string to the channel named channelId followed by a newline character. A trailing newline character is written unless the optional flag -nonewline is given. If channelId is omitted, the string is written to the standard output channel, stdout.

Newline characters in the output are translated by chan puts to platform-specific end-of-line sequences according to the currently configured value of the -translation option for the channel (for example, on PCs newlines are normally replaced with carriage-return-linefeed sequences; see chan configure above for details).

Tcl buffers output internally, so characters written with chan puts may not appear immediately on the output file or device; Tcl will normally delay output until the buffer is full or the channel is closed. You can force output to appear immediately with the chan flush command.

When the output buffer fills up, the chan puts command will normally block until all the buffered data has been accepted for
output by the operating system. If \textit{channelId} is in nonblocking mode then the \texttt{chan puts} command will not block even if the operating system cannot accept the data. Instead, Tcl continues to buffer the data and writes it in the background as fast as the underlying file or device can accept it. The application must use the Tcl event loop for nonblocking output to work; otherwise Tcl never finds out that the file or device is ready for more output data. It is possible for an arbitrarily large amount of data to be buffered for a channel in nonblocking mode, which could consume a large amount of memory. To avoid wasting memory, nonblocking I/O should normally be used in an event-driven fashion with the \texttt{chan event} command (do not invoke \texttt{chan puts} unless you have recently been notified via a file event that the channel is ready for more output data).

\texttt{chan read channelId ?numChars?}

\texttt{chan read ?-nonewline? channelId}

In the first form, the result will be the next \texttt{numChars} characters read from the channel named \texttt{channelId}; if \texttt{numChars} is omitted, all characters up to the point when the channel would signal a failure (whether an end-of-file, blocked or other error condition) are read. In the second form (i.e. when \texttt{numChars} has been omitted) the flag \texttt{-nonewline} may be given to indicate that any trailing newline in the string that has been read should be trimmed.

If \texttt{channelId} is in nonblocking mode, \texttt{chan read} may not read as many characters as requested: once all available input has been read, the command will return the data that is available rather than blocking for more input. If the channel is configured to use a multi-byte encoding, then there may actually be some bytes remaining in the internal buffers that do not form a complete character. These bytes will not be returned until a complete character is available or end-of-file is reached. The \texttt{-nonewline} switch is ignored if the command returns before reaching the end of the file.

\texttt{Chan read} translates end-of-line sequences in the input into newline characters according to the \texttt{-translation} option for the
channel (see chan configure above for a discussion on the ways in which chan configure will alter input).

When reading from a serial port, most applications should configure the serial port channel to be nonblocking, like this:

```
chan configure channelId -blocking 0.
```

Then chan read behaves much like described above. Note that most serial ports are comparatively slow; it is entirely possible to get a readable event for each character read from them. Care must be taken when using chan read on blocking serial ports:

```
chan read channelld numChars
```

In this form chan read blocks until numChars have been received from the serial port.

```
chan read channelld
```

In this form chan read blocks until the reception of the end-of-file character, see chan configure -eofchar. If there no end-of-file character has been configured for the channel, then chan read will block forever.

```
chan seek channelId offset ?origin?
```

Sets the current access position within the underlying data stream for the channel named channelId to be offset bytes relative to origin. Offset must be an integer (which may be negative) and origin must be one of the following:

```
start
```

The new access position will be offset bytes from the start of the underlying file or device.

```
current
```

The new access position will be offset bytes from the current access position; a negative offset moves the access position backwards in the underlying file or device.
The new access position will be offset bytes from the end of the file or device. A negative offset places the access position before the end of file, and a positive offset places the access position after the end of file.

The origin argument defaults to start.

**Chan seek** flushes all buffered output for the channel before the command returns, even if the channel is in nonblocking mode. It also discards any buffered and unread input. This command returns an empty string. An error occurs if this command is applied to channels whose underlying file or device does not support seeking.

Note that offset values are byte offsets, not character offsets. Both **chan seek** and **chan tell** operate in terms of bytes, not characters, unlike **chan read**.

**chan tell channelId**

Returns a number giving the current access position within the underlying data stream for the channel named **channelId**. This value returned is a byte offset that can be passed to **chan seek** in order to set the channel to a particular position. Note that this value is in terms of bytes, not characters like **chan read**. The value returned is -1 for channels that do not support seeking.

**chan truncate channelId ?length?**

Sets the byte length of the underlying data stream for the channel named **channelId** to be length (or to the current byte offset within the underlying data stream if length is omitted). The channel is flushed before truncation.

**EXAMPLE**

This opens a file using a known encoding (CP1252, a very common encoding on Windows), searches for a string, rewrites that part, and truncates the file after a further two lines.
set f [open somefile.txt r+]
chan configure $f -encoding cp1252
set offset 0

# Search for string "FOOBAR" in the file
while {[chan gets $f line] >= 0} {
    set idx [string first FOOBAR $line]
    if {$idx > -1} {
        # Found it; rewrite line
        chan seek $f [expr {$offset + $idx}]
        chan puts -nonewline $f BARFOO

        # Skip to end of following line, and truncate
        chan gets $f
        chan gets $f
        chan truncate $f

        # Stop searching the file now
        break
    }
}

# Save offset of start of next line for later
set offset [chan tell $f]
}
chan close $f

SEE ALSO

close, eof, fblocked, fconfigure, fcopy, file, fileevent, flush, gets, open, puts, read, seek, socket, tell, refchan

KEYWORDS

channel, input, output, events, offset
NAME
glob - Return names of files that match patterns
SYNOPSIS
DESCRIPTION
        -directory  directory
        -join
        -nocomplain
        -path  pathPrefix
        -tails
        -types  typeList
        -- ?
        *
        [chars]
        \x
        {a,b,...}
PORTABILITY ISSUES
EXAMPLES
SEE ALSO
KEYWORDS

NAME

glob - Return names of files that match patterns

SYNOPSIS

glob  ?switches?  pattern  ?pattern ...?

DESCRIPTION

This command performs file name “globbing” in a fashion similar to the
csh shell. It returns a list of the files whose names match any of the pattern arguments. No particular order is guaranteed in the list, so if a sorted list is required the caller should use lsort.

If the initial arguments to glob start with - then they are treated as switches. The following switches are currently supported:

-directory directory
  Search for files which match the given patterns starting in the given directory. This allows searching of directories whose name contains glob-sensitive characters without the need to quote such characters explicitly. This option may not be used in conjunction with -path, which is used to allow searching for complete file paths whose names may contain glob-sensitive characters.

-join
  The remaining pattern arguments are treated as a single pattern obtained by joining the arguments with directory separators.

-nocomplain
  Allows an empty list to be returned without error; without this switch an error is returned if the result list would be empty.

-path pathPrefix
  Search for files with the given pathPrefix where the rest of the name matches the given patterns. This allows searching for files with names similar to a given file (as opposed to a directory) even when the names contain glob-sensitive characters. This option may not be used in conjunction with -directory. For example, to find all files with the same root name as $path, but differing extensions, you should use glob -path [file rootname $path].* which will work even if $path contains numerous glob-sensitive characters.

-tails
  Only return the part of each file found which follows the last directory named in any -directory or -path path specification. Thus glob -tails -directory $dir * is equivalent to set pwd [pwd] ; cd $dir ; glob * ; cd $pwd. For -path specifications, the returned
names will include the last path segment, so `glob -tails -path [file rootname ~/foo.tex] .*` will return paths like `foo.aux foo.bib foo.tex` etc.

**-types typeList**

Only list files or directories which match `typeList`, where the items in the list have two forms. The first form is like the `-type` option of the Unix `find` command: `b` (block special file), `c` (character special file), `d` (directory), `f` (plain file), `l` (symbolic link), `p` (named pipe), or `s` (socket), where multiple types may be specified in the list. **Glob** will return all files which match at least one of the types given. Note that symbolic links will be returned both if `-types l` is given, or if the target of a link matches the requested type. So, a link to a directory will be returned if `-types d` was specified.

The second form specifies types where all the types given must match. These are `r`, `w`, `x` as file permissions, and `readonly`, `hidden` as special permission cases. On the Macintosh, MacOS types and creators are also supported, where any item which is four characters long is assumed to be a MacOS type (e.g. `TEXT`). Items which are of the form `{macintosh type XXXX}` or `{macintosh creator XXXX}` will match types or creators respectively. Unrecognized types, or specifications of multiple MacOS types/creators will signal an error.

The two forms may be mixed, so `-types {d f r w}` will find all regular files OR directories that have both read AND write permissions. The following are equivalent:

```plaintext
  glob -type d *
glob */
```

except that the first case doesn’t return the trailing “/” and is more platform independent.

--
Marks the end of switches. The argument following this one will be treated as a *pattern* even if it starts with a `-`.

The *pattern* arguments may contain any of the following special characters:

?  
Matches any single character.

*  
Matches any sequence of zero or more characters.

[chars]  
Matches any single character in *chars*. If *chars* contains a sequence of the form *a*-b then any character between *a* and *b* (inclusive) will match.

\x  
Matches the character *x*.

{a,b,...}  
Matches any of the strings *a*, *b*, etc.

On Unix, as with csh, a `.` at the beginning of a file's name or just after a `/*` must be matched explicitly or with a `[]` construct, unless the `-types hidden` flag is given (since `.` at the beginning of a file's name indicates that it is hidden). On other platforms, files beginning with a `.` are handled no differently to any others, except the special directories `.` and `..` which must be matched explicitly (this is to avoid a recursive pattern like `glob -join * * * *` from recursing up the directory hierarchy as well as down). In addition, all `/*` characters must be matched explicitly.

If the first character in a *pattern* is `~` then it refers to the home directory for the user whose name follows the `~`. If the `~` is followed immediately by `/*` then the value of the HOME environment variable is used.

The *glob* command differs from csh globbing in two ways. First, it does
not sort its result list (use the ls\texttt{sort} command if you want the list sorted). Second, \texttt{glob} only returns the names of files that actually exist; in csh no check for existence is made unless a pattern contains a \texttt{?}, \texttt{*}, or \texttt{[] constructs.

When the \texttt{glob} command returns relative paths whose filenames start with a tilde “~” (for example through \texttt{glob *} or \texttt{glob -tails}, the returned list will not quote the tilde with “./”. This means care must be taken if those names are later to be used with \texttt{file join}, to avoid them being interpreted as absolute paths pointing to a given user's home directory.

\section*{PORTABILITY ISSUES}

\textbf{Windows} For Windows UNC names, the servername and sharename components of the path may not contain \texttt{?}, \texttt{*}, or \texttt{[]} constructs. On Windows NT, if \texttt{pattern} is of the form “\texttt{~username@domain}”, it refers to the home directory of the user whose account information resides on the specified NT domain server. Otherwise, user account information is obtained from the local computer. On Windows 95 and 98, \texttt{glob} accepts patterns like “/.” and “../” for successively higher up parent directories.

Since the backslash character has a special meaning to the glob command, glob patterns containing Windows style path separators need special care. The pattern \texttt{C:\foo\*} is interpreted as \texttt{C:\foo\*} where \texttt{\*} will match the single character \texttt{f} and \texttt{\*} will match the single character \texttt{*} and will not be interpreted as a wildcard character. One solution to this problem is to use the Unix style forward slash as a path separator. Windows style paths can be converted to Unix style paths with the command \texttt{file join $path} (or \texttt{file normalize $path} in Tcl 8.4).

\section*{EXAMPLES}

Find all the Tcl files in the current directory:

\begin{verbatim}
   glob *.tcl
\end{verbatim}
Find all the Tcl files in the user's home directory, irrespective of what the current directory is:

```bash
glob -directory ~ *.tcl
```

Find all subdirectories of the current directory:

```bash
glob -type d *
```

Find all files whose name contains an “a”, a “b” or the sequence “cde”:

```bash
glob -type f *{a,b,cde}*
```

SEE ALSO

`file`

KEYWORDS

`exist`, `file`, `glob`, `pattern`
NAME
memory - Control Tcl memory debugging capabilities

SYNOPSIS
memory option ?arg arg ...?

DESCRIPTION
The memory command gives the Tcl developer control of Tcl's memory debugging capabilities. The memory command has several suboptions, which are described below. It is only available when Tcl has been compiled with memory debugging enabled (when TCL_MEM_DEBUG is defined at compile time), and after Tcl_InitMemory has been called.
memory active file
Write a list of all currently allocated memory to the specified file.

memory break_on_malloc count
After the count allocations have been performed, ckalloc outputs a message to this effect and that it is now attempting to enter the C debugger. Tcl will then issue a SIGINT signal against itself. If you are running Tcl under a C debugger, it should then enter the debugger command mode.

memory info
Returns a report containing the total allocations and frees since Tcl began, the current packets allocated (the current number of calls to ckalloc not met by a corresponding call to ckfree), the current bytes allocated, and the maximum number of packets and bytes allocated.

memory init [on|off]
Turn on or off the pre-initialization of all allocated memory with bogus bytes. Useful for detecting the use of uninitialized values.

memory objs file
Causes a list of all allocated Tcl_Obj values to be written to the specified file immediately, together with where they were allocated. Useful for checking for leaks of values.

memory onexit file
Causes a list of all allocated memory to be written to the specified file during the finalization of Tcl's memory subsystem. Useful for checking that memory is properly cleaned up during process exit.

memory tag string
Each packet of memory allocated by ckalloc can have associated with it a string-valued tag. In the lists of allocated memory generated by memory active and memory onexit, the tag for each packet is printed along with other information about the packet. The memory tag command sets the tag value for subsequent calls to ckalloc to be string.
**memory trace [on|off]**

Turns memory tracing on or off. When memory tracing is on, every call to `ckalloc` causes a line of trace information to be written to `stderr`, consisting of the word `ckalloc`, followed by the address returned, the amount of memory allocated, and the C filename and line number of the code performing the allocation. For example:

```
ckalloc 40e478 98 tclProc.c 1406
```

Calls to `ckfree` are traced in the same manner.

**memory trace_on_at_malloc count**

Enable memory tracing after `count` `ckalloc`S have been performed. For example, if you enter `memory trace_on_at_malloc 100`, after the 100th call to `ckalloc`, memory trace information will begin being displayed for all allocations and frees. Since there can be a lot of memory activity before a problem occurs, judicious use of this option can reduce the slowdown caused by tracing (and the amount of trace information produced), if you can identify a number of allocations that occur before the problem sets in. The current number of memory allocations that have occurred since Tcl started is printed on a guard zone failure.

**memory validate [on|off]**

Turns memory validation on or off. When memory validation is enabled, on every call to `ckalloc` or `ckfree`, the guard zones are checked for every piece of memory currently in existence that was allocated by `ckalloc`. This has a large performance impact and should only be used when overwrite problems are strongly suspected. The advantage of enabling memory validation is that a guard zone overwrite can be detected on the first call to `ckalloc` or `ckfree` after the overwrite occurred, rather than when the specific memory with the overwritten guard zone(s) is freed, which may occur long after the overwrite occurred.
ckalloc, ckfree, Tcl.ValidateAllMemory, Tcl.DumpActiveMemory, TCL_MEM_DEBUG

KEYWORDS

memory, debug

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1992-1999 by Karl Lehenbauer and Mark Diekhans
Copyright © 2000 by Scriptics Corporation.
NAME

set - Read and write variables

SYNOPSIS

set varName ?value?

DESCRIPTION

Returns the value of variable varName. If value is specified, then set the value of varName to value, creating a new variable if one does not already exist, and return its value. If varName contains an open parenthesis and ends with a close parenthesis, then it refers to an array element: the characters before the first open parenthesis are the name of the array, and the characters between the parentheses are the index within the array. Otherwise varName refers to a scalar variable.

If varName includes namespace qualifiers (in the array name if it refers to an array element), or if varName is unqualified (does not include the names of any containing namespaces) but no procedure is active, varName refers to a namespace variable resolved according to the rules described under NAME RESOLUTION in the namespace manual page.

If a procedure is active and varName is unqualified, then varName refers to a parameter or local variable of the procedure, unless varName was declared to resolve differently through one of the global, variable or upvar commands.

EXAMPLES
Store a random number in the variable $r$:

```tcl
set r [expr {rand()}]
```

Store a short message in an array element:

```tcl
set anAry(msg) "Hello, World!"
```

Store a short message in an array element specified by a variable:

```tcl
set elemName "msg"
set anAry($elemName) "Hello, World!"
```

Copy a value into the variable `out` from a variable whose name is stored in the `vbl` (note that it is often easier to use arrays in practice instead of doing double-dereferencing):

```tcl
set in0 "small random"
set in1 "large random"
set vbl in[expr {rand() >= 0.5}]
set out [set $vbl]
```

SEE ALSO

`expr`, `global`, `namespace`, `proc`, `trace`, `unset`, `upvar`, `variable`

KEYWORDS

`read`, `write`, `variable`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
NAME

upvar - Create link to variable in a different stack frame

SYNOPSIS

```
upvar ?level? otherVar myVar ?otherVar myVar ...?
```

DESCRIPTION

This command arranges for one or more local variables in the current procedure to refer to variables in an enclosing procedure call or to global variables. Level may have any of the forms permitted for the `uplevel` command, and may be omitted if the first letter of the first otherVar is not # or a digit (it defaults to 1). For each otherVar argument, `upvar` makes the variable by that name in the procedure frame given by level (or at global level, if level is #0) accessible in the current procedure by the name given in the corresponding myVar argument. The variable named by otherVar need not exist at the time of the call; it will be created the first time myVar is referenced, just like an ordinary variable. There must not exist a variable by the name myVar at the time `upvar` is invoked. MyVar is always treated as the name of a variable, not an array element. An error is returned if the name looks like an array element, such as a(b). OtherVar may refer to a scalar variable, an array, or an array element. Upvar returns an empty string.

The `upvar` command simplifies the implementation of call-by-name procedure calling and also makes it easier to build new control constructs as Tcl procedures. For example, consider the following procedure:
If `add2` is invoked with an argument giving the name of a variable, it adds two to the value of that variable. Although `add2` could have been implemented using `uplevel` instead of `upvar`, `upvar` makes it simpler for `add2` to access the variable in the caller's procedure frame.

`namespace eval` is another way (besides procedure calls) that the Tcl naming context can change. It adds a call frame to the stack to represent the namespace context. This means each `namespace eval` command counts as another call level for `uplevel` and `upvar` commands. For example, `info level 1` will return a list describing a command that is either the outermost procedure call or the outermost `namespace eval` command. Also, `uplevel #0` evaluates a script at top-level in the outermost namespace (the global namespace).

If an upvar variable is unset (e.g. `x` in `add2` above), the `unset` operation affects the variable it is linked to, not the upvar variable. There is no way to unset an upvar variable except by exiting the procedure in which it is defined. However, it is possible to retarget an upvar variable by executing another `upvar` command.

**TRACES AND UPVAR**

Upvar interacts with traces in a straightforward but possibly unexpected manner. If a variable trace is defined on `otherVar`, that trace will be triggered by actions involving `myVar`. However, the trace procedure will be passed the name of `myVar`, rather than the name of `otherVar`. Thus, the output of the following code will be "`localVar`" rather than "`originalVar`":

```tcl
proc traceproc { name index op } {
    ```
puts $name
} proc setByUpvar { name value } {
  upvar $name localVar
  set localVar $value
}
set originalVar 1
trace variable originalVar w traceproc
setByUpvar originalVar 2

If otherVar refers to an element of an array, then variable traces set for the entire array will not be invoked when myVar is accessed (but traces on the particular element will still be invoked). In particular, if the array is env, then changes made to myVar will not be passed to subprocesses correctly.

**EXAMPLE**

A **decr** command that works like **incr** except it subtracts the value from the variable instead of adding it:

```
proc decr {varName {decrement 1}} {
  upvar 1 $varName var
  incr var [expr {-$decrement}]
}
```

**SEE ALSO**

global, namespace, uplevel, variable

**KEYWORDS**

context, frame, global, level, namespace, procedure, variable

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
NAME
clock - Obtain and manipulate dates and times

SYNOPSIS

DESCRIPTION

clock add timeVal ?count unit...? ?-option value?
clock clicks ?-option?
clock format timeVal ?-option value...?
clock microseconds

clock milliseconds

clock scan inputString ?-option value...?
clock seconds

PARAMETERS

count
timeVal
unit

OPTIONS

-base time
-format format
-gmt boolean
-locale localeName
-timezone zoneName

CLOCK ARITHMETIC

HIGH RESOLUTION TIMERS

FORMATTING TIMES

SCANNING TIMES

FORMAT GROUPS

%a
%A
%b
%B
%c
NAME
clock - Obtain and manipulate dates and times

SYNOPSIS

package require **Tcl 8.5**
clock add timeVal ?count unit...? ?-option value?
clock clicks ?-option?
clock format timeVal ?-option value...?
clock microseconds
clock milliseconds
clock scan inputString ?-option value...?
clock seconds

DESCRIPTION
The **clock** command performs several operations that obtain and manipulate values that represent times. The command supports several subcommands that determine what action is carried out by the command.

**clock add** `timeVal ?count unit...? ?-option value?`  
Adds a (possibly negative) offset to a time that is expressed as an integer number of seconds. See **CLOCK ARITHMETIC** for a full description.

**clock clicks** `?-option?`  
If no `-option` argument is supplied, returns a high-resolution time value as a system-dependent integer value. The unit of the value is system-dependent but should be the highest resolution clock available on the system such as a CPU cycle counter. See **HIGH RESOLUTION TIMERS** for a full description.

If the `-option` argument is `-milliseconds`, then the command is synonymous with **clock milliseconds** (see below). This usage is obsolete, and **clock milliseconds** is to be considered the preferred way of obtaining a count of milliseconds.

If the `-option` argument is `-microseconds`, then the command is synonymous with **clock microseconds** (see below). This usage is obsolete, and **clock microseconds** is to be considered the preferred way of obtaining a count of microseconds.

**clock format** `timeVal ?-option value...?`  
Formats a time that is expressed as an integer number of seconds into a format intended for consumption by users or external programs. See **FORMATTING TIMES** for a full description.

**clock microseconds**  
Returns the current time as an integer number of microseconds. See **HIGH RESOLUTION TIMERS** for a full description.

**clock milliseconds**  
Returns the current time as an integer number of milliseconds. See
**HIGH RESOLUTION TIMERS** for a full description.

**clock scan** *inputString ?-option value...?
Scans a time that is expressed as a character string and produces an integer number of seconds. See **SCANNING TIMES** for a full description.

**clock seconds**
Returns the current time as an integer number of seconds.

---

**PARAMETERS**

**count**
An integer representing a count of some unit of time. See **CLOCK ARITHMETIC** for the details.

**timeVal**
An integer value passed to the **clock** command that represents an absolute time as a number of seconds from the **epoch time** of 1 January 1970, 00:00 UTC. Note that the count of seconds does not include any leap seconds; seconds are counted as if each UTC day has exactly 86400 seconds. Tcl responds to leap seconds by speeding or slowing its clock by a tiny fraction for some minutes until it is back in sync with UTC; its data model does not represent minutes that have 59 or 61 seconds.

**unit**
One of the words, **seconds**, **minutes**, **hours**, **days**, **weeks**, **months**, or **years**, or any unique prefix of such a word. Used in conjunction with **count** to identify an interval of time, for example, 3 **seconds** or 1 **year**.

---

**OPTIONS**

**-base time**
Specifies that any relative times present in a **clock scan** command are to be given relative to **time**. **time** must be expressed as a count of nominal seconds from the epoch time of 1 January 1970, 00:00...
-format format
Specifies the desired output format for clock format or the expected input format for clock scan. The format string consists of any number of characters other than the per-cent sign (“%”) interspersed with any number of format groups, which are two-character sequences beginning with the per-cent sign. The permissible format groups, and their interpretation, are described under FORMAT GROUPS.

On clock format, the default format is

```
%a %b %d %H:%M:%S %z %Y
```

On clock scan, the lack of a -format option indicates that a “free format scan” is requested; see FREE FORM SCAN for a description of what happens.

-gmt boolean
If boolean is true, specifies that a time specified to clock add, clock format or clock scan should be processed in UTC. If boolean is false, the processing defaults to the local time zone. This usage is obsolete; the correct current usage is to specify the UTC time zone with “-timezone :UTC” or any of the equivalent ways to specify it.

-locale localeName
Specifies that locale-dependent scanning and formatting (and date arithmetic for dates preceding the adoption of the Gregorian calendar) is to be done in the locale identified by localeName. The locale name may be any of the locales acceptable to the msgcat package, or it may be the special name system, which represents the current locale of the process, or the null string, which represents Tcl's default locale.
The effect of locale on scanning and formatting is discussed in the descriptions of the individual format groups under **FORMAT GROUPS**. The effect of locale on clock arithmetic is discussed under **CLOCK ARITHMETIC**.

**-timezone zoneName**

Specifies that clock arithmetic, formatting, and scanning are to be done according to the rules for the time zone specified by `zoneName`. The permissible values, and their interpretation, are discussed under **TIME ZONES**. On subcommands that expect a **-timezone** argument, the default is to use the *current time zone*. The current time zone is determined, in order of preference, by:

1. the environment variable `TCL_TZ`.
2. the environment variable `TZ`.
3. on Windows systems, the time zone settings from the Control Panel.

If none of these is present, the C **localtime** and **mktime** functions are used to attempt to convert times between local and Greenwich. On 32-bit systems, this approach is likely to have bugs, particularly for times that lie outside the window (approximately the years 1902 to 2037) that can be represented in a 32-bit integer.

**CLOCK ARITHMETIC**

The **clock add** command performs clock arithmetic on a value (expressed as nominal seconds from the epoch time of 1 January 1970, 00:00 UTC) given as its first argument. The remaining arguments (other than the possible **-timezone**, **-locale** and **-gmt** options) are integers and keywords in alternation, where the keywords are chosen from **seconds**, **minutes**, **hours**, **days**, **weeks**, **months**, or **years**, or any unique prefix of such a word.
Addition of seconds, minutes and hours is fairly straightforward; the given time increment (times sixty for minutes, or 3600 for hours) is simply added to the *timeVal* given to the **clock add** command. The result is interpreted as a nominal number of seconds from the Epoch.

Surprising results may be obtained when crossing a point at which a leap second is inserted or removed; the **clock add** command simply ignores leap seconds and therefore assumes that times come in sequence, 23:59:58, 23:59:59, 00:00:00. (This assumption is handled by the fact that Tcl's model of time reacts to leap seconds by speeding or slowing the clock by a minuscule amount until Tcl's time is back in step with the world.

The fact that adding and subtracting hours is defined in terms of absolute time means that it will add fixed amounts of time in time zones that observe summer time (Daylight Saving Time). For example, the following code sets the value of *x* to **04:00:00** because the clock has changed in the interval in question.

```tcl
set s [clock scan {2004-10-30 05:00:00} \
    -format {%Y-%m-%d %H:%M:%S} \n    -timezone :America/New_York]
set a [clock add $s 24 hours -timezone :America/New_\nset x [clock format $a \n    -format {%H:%M:%S} -timezone :America/New
```

Adding and subtracting days and weeks is accomplished by converting the given time to a calendar day and time of day in the appropriate time zone and locale. The requisite number of days (weeks are converted to days by multiplying by seven) is added to the calendar day, and the date and time are then converted back to a count of seconds from the epoch time.

Adding and subtracting a given number of days across the point that the time changes at the start or end of summer time (Daylight Saving Time)
results in the *same local time* on the day in question. For instance, the following code sets the value of \( x \) to 05:00:00.

\[
\text{set } s \ [\text{clock scan } \{2004-10-30 \ 05:00:00\}] \\
\text{set } a \ [\text{clock add } s \ 1 \ \text{day}] \\
\text{set } x \ [\text{clock format } a]
\]

In cases of ambiguity, where the same local time happens twice on the same day, the earlier time is used. In cases where the conversion yields an impossible time (for instance, 02:30 during the Spring Daylight Saving Time change using US rules), the time is converted as if the clock had not changed. Thus, the following code will set the value of \( x \) to 03:30:00.

\[
\text{set } s \ [\text{clock scan } \{2004-04-03 \ 02:30:00\}] \\
\text{set } a \ [\text{clock add } s \ 1 \ \text{day}] \\
\text{set } x \ [\text{clock format } a]
\]

Adding a given number of days or weeks works correctly across the conversion between the Julian and Gregorian calendars; the omitted days are skipped. The following code sets \( z \) to 1752-09-14.

\[
\text{set } x \ [\text{clock scan } \{1752-09-02\}] \\
\text{set } y \ [\text{clock add } x \ 1 \ \text{day}] \\
\text{set } z \ [\text{clock format } y]
\]
In the bizarre case that adding the given number of days yields a date that does not exist because it falls within the dropped days of the Julian-to-Gregorian conversion, the date is converted as if it was on the Julian calendar.

Adding a number of months, or a number of years, is similar; it converts the given time to a calendar date and time of day. It then adds the requisite number of months or years, and reconverts the resulting date and time of day to an absolute time.

If the resulting date is impossible because the month has too few days (for example, when adding 1 month to 31 January), the last day of the month is substituted. Thus, adding 1 month to 31 January will result in 28 February in a common year or 29 February in a leap year.

The rules for handling anomalies relating to summer time and to the Gregorian calendar are the same when adding/subtracting months and years as they are when adding/subtracting days and weeks.

If multiple count unit pairs are present on the command, they are evaluated consecutively, from left to right.

### HIGH RESOLUTION TIMERS

Most of the subcommands supported by the clock command deal with times represented as a count of seconds from the epoch time, and this is the representation that clock seconds returns. There are three exceptions, which are all intended for use where higher-resolution times are required. clock milliseconds returns the count of milliseconds from the epoch time, and clock microseconds returns the count of microseconds from the epoch time. In addition, there is a clock clicks command that returns a platform-dependent high-resolution timer. Unlike clock seconds and clock milliseconds, the value of clock clicks is not guaranteed to be tied to any fixed epoch; it is simply intended to be the most precise interval timer available, and is intended only for relative timing studies such as benchmarks.

### FORMATTING TIMES
The **clock format** command produces times for display to a user or writing to an external medium. The command accepts times that are expressed in seconds from the epoch time of 1 January 1970, 00:00 UTC, as returned by **clock seconds**, **clock scan**, **clock add**, **file atime** or **file mtime**.

If a **-format** option is present, the following argument is a string that specifies how the date and time are to be formatted. The string consists of any number of characters other than the per-cent sign (“%”) interspersed with any number of **format groups**, which are two-character sequences beginning with the per-cent sign. The permissible format groups, and their interpretation, are described under **FORMAT GROUPS**.

If a **-timezone** option is present, the following argument is a string that specifies the time zone in which the date and time are to be formatted. As an alternative to “**-timezone :UTC**”, the obsolete usage “**-gmt true**” may be used. See **TIME ZONES** for the permissible variants for the time zone.

If a **-locale** option is present, the following argument is a string that specifies the locale in which the time is to be formatted, in the same format that is used for the **msgcat** package. Note that the default, if **-locale** is not specified, is the root locale `{}` rather than the current locale. The current locale may be obtained by using **-locale current**. In addition, some platforms support a **system** locale that reflects the user’s current choices. For instance, on Windows, the format that the user has selected from dates and times in the Control Panel can be obtained by using the **system** locale. On platforms that do not define a user selection of date and time formats separate from **LC_TIME**, **-locale system** is synonymous with **-locale current**.

### SCANNING TIMES

The **clock scan** command accepts times that are formatted as strings and converts them to counts of seconds from the epoch time of 1 January 1970, 00:00 UTC. It normally takes a **-format** option that is followed by a string describing the expected format of the input. (See
**FREE FORM SCAN** for the effect of **clock scan** without such an argument.) The string consists of any number of characters other than the per-cent sign (“%”), interspersed with any number of **format groups**, which are two-character sequences beginning with the per-cent sign. The permissible format groups, and their interpretation, are described under **FORMAT GROUPS**.

If a **-timezone** option is present, the following argument is a string that specifies the time zone in which the date and time are to be interpreted. As an alternative to **-timezone :UTC**, the obsolete usage **-gmt true** may be used. See **TIME ZONES** for the permissible variants for the time zone.

If a **-locale** option is present, the following argument is a string that specifies the locale in which the time is to be interpreted, in the same format that is used for the **msgcat** package. Note that the default, if **-locale** is not specified, is the root locale `{}` rather than the current locale. The current locale may be obtained by using **-locale current**. In addition, some platforms support a **system** locale that reflects the user's current choices. For instance, on Windows, the format that the user has selected from dates and times in the Control Panel can be obtained by using the **system** locale. On platforms that do not define a user selection of date and time formats separate from **LC_TIME**, **-locale system** is synonymous with **-locale current**.

If a **-base** option is present, the following argument is a time (expressed in seconds from the epoch time) that is used as a **base time** for interpreting relative times. If no **-base** option is present, the base time is the current time.

Scanning of times in fixed format works by determining three things: the date, the time of day, and the time zone. These three are then combined into a point in time, which is returned as the number of seconds from the epoch.

Before scanning begins, the format string is preprocessed to replace `%c`, `%Ec`, `%x`, `%Ex`, `%X`, `%Ex`, `%r`, `%R`, `%T`, `%D`, `%EY` and `%+` format groups with counterparts that are appropriate to the current locale and
contain none of the above groups. For instance, %D will (in the en_US locale) be replaced with %m/%d/%Y.

The date is determined according to the fields that are present in the preprocessed format string. In order of preference:

[1] If the string contains a %s format group, representing seconds from the epoch, that group is used to determine the date.

[2] If the string contains a %J format group, representing the Julian Day Number, that group is used to determine the date.

[3] If the string contains a complete set of format groups specifying century, year, month, and day of month; century, year, and day of year; or ISO8601 fiscal year, week of year, and day of week; those groups are combined and used to determine the date. If more than one complete set is present, the one at the rightmost position in the string is used.

[4] If the string lacks a century but contains a set of format groups specifying year of century, month and day of month; year of century and day of year; or two-digit ISO8601 fiscal year, week of year, and day of week; those groups are combined and used to determine the date. If more than one complete set is present, the one at the rightmost position in the string is used. The year is presumed to lie in the range 1938 to 2037 inclusive.

[5] If the string entirely lacks any specification for the year (or contains the year only on the locale's alternative calendar) and contains a set of format groups specifying month and day of month, day of year, or week of year and day of week, those groups are combined and used to determine the date. If more than one complete set is present, the one at the rightmost position in the string is used. The
year is determined by interpreting the base time in the given time zone.

[6] If the string contains none of the above sets, but has a day of the month or day of the week, the day of the month or day of the week are used to determine the date by interpreting the base time in the given time zone and returning the given day of the current week or month. (The week runs from Monday to Sunday, ISO8601-fashion.) If both day of month and day of week are present, the day of the month takes priority.

[7] If none of the above rules results in a usable date, the date of the base time in the given time zone is used.

The time is also determined according to the fields that are present in the preprocessed format string. In order of preference:

[1] If the string contains a %s format group, representing seconds from the epoch, that group determines the time of day.

[2] If the string contains either an hour on the 24-hour clock or an hour on the 12-hour clock plus an AM/PM indicator, that hour determines the hour of the day. If the string further contains a group specifying the minute of the hour, that group combines with the hour. If the string further contains a group specifying the second of the minute, that group combines with the hour and minute.

[3] If the string contains neither a %s format group nor a group specifying the hour of the day, then midnight (00:00, the start of the given date) is used. The time zone is determined by either the -timezone or -gmt options, or by using the current time zone.

If a format string lacks a %z or %Z format group, it is possible for the
time to be ambiguous because it appears twice in the same day, once without and once with Daylight Saving Time. If this situation occurs, the first occurrence of the time is chosen. (For this reason, it is wise to have the input string contain the time zone when converting local times. This caveat does not apply to UTC times.)

**FORMAT GROUPS**

The following format groups are recognized by the `clock scan` and `clock format` commands.

**%a**

On output, receives an abbreviation (e.g., `Mon`) for the day of the week in the given locale. On input, matches the name of the day of the week in the given locale (in either abbreviated or full form, or any unique prefix of either form).

**%A**

On output, receives the full name (e.g., `Monday`) of the day of the week in the given locale. On input, matches the name of the day of the week in the given locale (in either abbreviated or full form, or any unique prefix of either form).

**%b**

On output, receives an abbreviation (e.g., `Jan`) for the name of the month in the given locale. On input, matches the name of the month in the given locale (in either abbreviated or full form, or any unique prefix of either form).

**%B**

On output, receives the full name (e.g., `January`) of the month in the given locale. On input, matches the name of the month in the given locale (in either abbreviated or full form, or any unique prefix of either form).

**%c**

On output, receives a localized representation of date and time of day; the localized representation is expected to use the Gregorian
calendar. On input, matches whatever \%c produces.

\%C
On output, receives the number of the century in Indo-Arabic numerals. On input, matches one or two digits, possibly with leading whitespace, that are expected to be the number of the century.

\%d
On output, produces the number of the day of the month, as two decimal digits. On input, matches one or two digits, possibly with leading whitespace, that are expected to be the number of the day of the month.

\%D
This format group is synonymous with \%m/%d/%Y. It should be used only in exchanging data within the en_US locale, since other locales typically do not use this order for the fields of the date.

\%e
On output, produces the number of the day of the month, as one or two decimal digits (with a leading blank for one-digit dates). On input, matches one or two digits, possibly with leading whitespace, that are expected to be the number of the day of the month.

\%Ec
On output, produces a locale-dependent representation of the date and time of day in the locale's alternative calendar. On input, matches whatever \%Ec produces. The locale's alternative calendar need not be the Gregorian calendar.

\%EC
On output, produces a locale-dependent name of an era in the locale's alternative calendar. On input, matches the name of the era or any unique prefix.

\%EE
On output, produces the string B.C.E. or C.E., or a string of the
same meaning in the locale, to indicate whether %Y refers to years before or after Year 1 of the Common Era. On input, accepts the string B.C.E., B.C., C.E., A.D., or the abbreviation appropriate to the current locale, and uses it to fix whether %Y refers to years before or after Year 1 of the Common Era.

%Ex
On output, produces a locale-dependent representation of the date in the locale's alternative calendar. On input, matches whatever %Ex produces. The locale's alternative calendar need not be the Gregorian calendar.

%EX
On output, produces a locale-dependent representation of the time of day in the locale's alternative numerals. On input, matches whatever %EX produces.

%Ey
On output, produces a locale-dependent number of the year of the era in the locale's alternative calendar and numerals. On input, matches such a number.

%EY
On output, produces a representation of the year in the locale's alternative calendar and numerals. On input, matches what %EY produces. Often synonymous with %EC%Ey.

%g
On output, produces a two-digit year number suitable for use with the week-based ISO8601 calendar; that is, the year number corresponds to the week number produced by %V. On input, accepts such a two-digit year number, possibly with leading whitespace.

%G
On output, produces a four-digit year number suitable for use with the week-based ISO8601 calendar; that is, the year number corresponds to the week number produced by %V. On input,
accepts such a four-digit year number, possibly with leading whitespace.

%h
This format group is synonymous with %b.

%H
On output, produces a two-digit number giving the hour of the day (00-23) on a 24-hour clock. On input, accepts such a number.

%I
On output, produces a two-digit number giving the hour of the day (12-11) on a 12-hour clock. On input, accepts such a number.

%J
On output, produces a three-digit number giving the day of the year (001-366). On input, accepts such a number.

%j
On input, accepts such a number.

%J
On output, produces a string of digits giving the Julian Day Number. On input, accepts a string of digits and interprets it as a Julian Day Number. The Julian Day Number is a count of the number of calendar days that have elapsed since 1 January, 4713 BCE of the proleptic Julian calendar. The epoch time of 1 January 1970 corresponds to Julian Day Number 2440588.

%k
On output, produces a one- or two-digit number giving the hour of the day (0-23) on a 24-hour clock. On input, accepts such a number.

%l
On output, produces a one- or two-digit number giving the hour of the day (12-11) on a 12-hour clock. On input, accepts such a number.

%m
On output, produces the number of the month (01-12) with exactly
two digits. On input, accepts two digits and interprets them as the number of the month.

%M
On output, produces the number of the minute of the hour (00-59) with exactly two digits. On input, accepts two digits and interprets them as the number of the minute of the hour.

%N
On output, produces the number of the month (1-12) with one or two digits, and a leading blank for one-digit dates. On input, accepts one or two digits, possibly with leading whitespace, and interprets them as the number of the month.

%Od, %Oe, %OH, %Ol, %Ok, %Ol, %Om, %OM, %OS, %Ou, %Ow, %Oy
All of these format groups are synonymous with their counterparts without the “O”, except that the string is produced and parsed in the locale-dependent alternative alternative numerals.

%p
On output, produces an indicator for the part of the day, AM or PM, appropriate to the given locale. If the script of the given locale supports multiple letterforms, lowercase is preferred. On input, matches the representation AM or PM in the given locale, in either case.

%P
On output, produces an indicator for the part of the day, am or pm, appropriate to the given locale. If the script of the given locale supports multiple letterforms, uppercase is preferred. On input, matches the representation AM or PM in the given locale, in either case.

%Q
This format group is reserved for internal use within the Tcl library.

%r
On output, produces a locale-dependent time of day representation on a 12-hour clock. On input, accepts whatever %r produces.

%R
On output, produces a locale-dependent time of day representation on a 24-hour clock. On input, accepts whatever %R produces.

%s
On output, simply formats the timeVal argument as a decimal integer and inserts it into the output string. On input, accepts a decimal integer and uses is as the time value without any further processing. Since %s uniquely determines a point in time, it overrides all other input formats.

%S
On output, produces a two-digit number of the second of the minute (00-59). On input, accepts two digits and uses them as the second of the minute.

%t
On output, produces a TAB character. On input, matches a TAB character.

%T
Synonymous with %H:%M:%S.

%u
On output, produces the number of the day of the week (1 → Monday, 7 → Sunday). On input, accepts a single digit and interprets it as the day of the week. Sunday may be either 0 or 7.

%U
On output, produces the ordinal number of the week of the year (00-53). The first Sunday of the year is the first day of week 01. On input accepts two digits which are otherwise ignored. This format group is never used in determining an input date. This interpretation of the week of the year was once common in US banking but is now largely obsolete. See %V for the ISO8601 week
number.

%V
On output, produces the number of the ISO8601 week as a two
digit number (01-53). Week 01 is the week containing January 4; or
the first week of the year containing at least 4 days; or the week
containing the first Thursday of the year (the three statements are
equivalent). Each week begins on a Monday. On input, accepts the
ISO8601 week number.

%w
On output, produces the ordinal number of the day of the week
(Sunday==0; Saturday==6). On input, accepts a single digit and
interprets it as the day of the week; Sunday may be represented as
either 0 or 7. Note that %w is not the ISO8601 weekday number,
which is produced and accepted by %u.

%W
On output, produces a week number (00-53) within the year; week
01 begins on the first Monday of the year. On input, accepts two
digits, which are otherwise ignored. This format group is never
used in determining an input date. It is not the ISO8601 week
number; that week is produced and accepted by %V.

%x
On output, produces the date in a locale-dependent representation.
On input, accepts whatever %x produces and is used to determine
calendar date.

%X
On output, produces the time of day in a locale-dependent
representation. On input, accepts whatever %X produces and is
used to determine time of day.

%y
On output, produces the two-digit year of the century. On input,
accepts two digits, and is used to determine calendar date. The
date is presumed to lie between 1938 and 2037 inclusive. Note that
%y does not yield a year appropriate for use with the ISO8601 week number %V; programs should use %g for that purpose.

%Y
On output, produces the four-digit calendar year. On input, accepts four digits and may be used to determine calendar date. Note that %Y does not yield a year appropriate for use with the ISO8601 week number %V; programs should use %G for that purpose.

%z
On output, produces the current time zone, expressed in hours and minutes east (+hhmm) or west (-hhmm) of Greenwich. On input, accepts a time zone specifier (see TIME ZONES below) that will be used to determine the time zone.

%Z
On output, produces the current time zone's name, possibly translated to the given locale. On input, accepts a time zone specifier (see TIME ZONES below) that will be used to determine the time zone. This option should, in general, be used on input only when parsing RFC822 dates. Other uses are fraught with ambiguity; for instance, the string BST may represent British Summer Time or Brazilian Standard Time. It is recommended that date/time strings for use by computers use numeric time zones instead.

%%% 
On output, produces a literal “%” character. On input, matches a literal “%” character.

%+
Synonymous with “%a %b %e %H:%M:%S %Z %Y”.

**TIME ZONES**

When the clock command is processing a local time, it has several possible sources for the time zone to use. In order of preference, they are:
[1] A time zone specified inside a string being parsed and matched by a \%z or \%Z format group.

[2] A time zone specified with the -timezone option to the clock command (or, equivalently, by -gmt 1).


[4] A time zone specified in an environment variable TZ.

[5] The local time zone from the Control Panel on Windows systems.

[6] The C library's idea of the local time zone, as defined by the mktime and localtime functions.

In case [1] only, the string is tested to see if it is one of the strings:

<table>
<thead>
<tr>
<th>gmt</th>
<th>ut</th>
<th>utc</th>
<th>bst</th>
<th>wet</th>
<th>wat</th>
<th>at</th>
</tr>
</thead>
<tbody>
<tr>
<td>nft</td>
<td>nst</td>
<td>ndt</td>
<td>ast</td>
<td>adt</td>
<td>est</td>
<td>edt</td>
</tr>
<tr>
<td>cst</td>
<td>cdt</td>
<td>mst</td>
<td>mdt</td>
<td>pst</td>
<td>pdt</td>
<td>yst</td>
</tr>
<tr>
<td>ydt</td>
<td>hst</td>
<td>hdt</td>
<td>cat</td>
<td>ahst</td>
<td>nt</td>
<td>idl</td>
</tr>
<tr>
<td>cet</td>
<td>cest</td>
<td>met</td>
<td>mewt</td>
<td>mest</td>
<td>swt</td>
<td>sst</td>
</tr>
<tr>
<td>eet</td>
<td>eest</td>
<td>bt</td>
<td>it</td>
<td>zp4</td>
<td>zp5</td>
<td>ist</td>
</tr>
<tr>
<td>zp6</td>
<td>wast</td>
<td>wadt</td>
<td>jt</td>
<td>cct</td>
<td>jst</td>
<td>cas</td>
</tr>
<tr>
<td>cadt</td>
<td>east</td>
<td>eadt</td>
<td>gst</td>
<td>nzt</td>
<td>nzst</td>
<td>nzd</td>
</tr>
<tr>
<td>idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If it is a string in the above list, it designates a known time zone, and is interpreted as such.
For time zones in case [1] that do not match any of the above strings, and always for cases [2]-[6], the following rules apply.

If the time zone begins with a colon, it is one of a standardized list of names like :America/New_York that give the rules for various locales. A complete list of the location names is too lengthy to be listed here. On most Tcl installations, the definitions of the locations are to be found in named files in the directory “/no_backup/tools/lib/tcl8.5/clock/tzdata”. On some Unix systems, these files are omitted, and the definitions are instead obtained from system files in “/usr/share/zoneinfo”, “/usr/share/lib/zoneinfo” or “/usr/local/etc/zoneinfo”. As a special case, the name :localtime refers to the local time zone as defined by the C library.

A time zone string consisting of a plus or minus sign followed by four or six decimal digits is interpreted as an offset in hours, minutes, and seconds (if six digits are present) from UTC. The plus sign denotes a sign east of Greenwich; the minus sign one west of Greenwich.

A time zone string conforming to the Posix specification of the TZ environment variable will be recognized. The specification may be found at http://www.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap08.html.

Any other time zone string is processed by prefixing a colon and attempting to use it as a location name, as above.

**LOCALIZATION**

Developers wishing to localize the date and time formatting and parsing are referred to http://tip.tcl.tk/173 for a specification.

**FREE FORM SCAN**

If the clock scan command is invoked without a -format option, then it requests a free-form scan. This form of scan is deprecated. The reason for the deprecation is that there are too many ambiguities. (Does the string “2000” represent a year, a time of day, or a quantity?) No set of
rules for interpreting free-form dates and times has been found to give unsurprising results in all cases.

If free-form scan is used, only the -base and -gmt options are accepted. The -timezone and -locale options will result in an error if -format is not supplied.

For the benefit of users who need to understand legacy code that uses free-form scan, the documentation for how free-form scan interprets a string is included here:

If only a time is specified, the current date is assumed. If the inputString does not contain a time zone mnemonic, the local time zone is assumed, unless the -gmt argument is true, in which case the clock value is calculated assuming that the specified time is relative to Greenwich Mean Time. -gmt, if specified, affects only the computed time value; it does not impact the interpretation of -base.

If the -base flag is specified, the next argument should contain an integer clock value. Only the date in this value is used, not the time. This is useful for determining the time on a specific day or doing other date-relative conversions.

The inputString argument consists of zero or more specifications of the following form:

time
A time of day, which is of the form: hh?:mm?:ss?? ?meridian? ?zone? or hhmm ?meridian? ?zone? If no meridian is specified, hh is interpreted on a 24-hour clock.

date
A specific month and day with optional year. The acceptable formats are “mm/dd/yyyy”, “monthname dd?, yy?”, “day, dd monthname ?yy?”, “dd monthname yy”, “?CC?yymmd”, and “dd-monthname-?CC?yy”. The default year is the current year. If the year is less than 100, we treat the years 00-68 as 2000-2068 and the years 69-99 as 1969-1999. Not all platforms can represent
the years 38-70, so an error may result if these years are used.

ISO 8601 point-in-time
An ISO 8601 point-in-time specification, such as "CCyymmddThhmmss," where T is the literal "T", "CCyymmdd hhmmss", or "CCyymmddThh:mm:ss". Note that only these three formats are accepted. The command does not accept the full range of point-in-time specifications specified in ISO8601. Other formats can be recognized by giving an explicit -format option to the clock scan command.

relative time
A specification relative to the current time. The format is number unit. Acceptable units are year, fortnight, month, week, day, hour, minute (or min), and second (or sec). The unit can be specified as a singular or plural, as in 3 weeks. These modifiers may also be specified: tomorrow, yesterday, today, now, last, this, next, ago.

The actual date is calculated according to the following steps.

First, any absolute date and/or time is processed and converted. Using that time as the base, day-of-week specifications are added. Next, relative specifications are used. If a date or day is specified, and no absolute or relative time is given, midnight is used. Finally, a correction is applied so that the correct hour of the day is produced after allowing for daylight savings time differences and the correct date is given when going from the end of a long month to a short month.

SEE ALSO
msgcat

KEYWORDS
clock, date, time

COPYRIGHT
NAME

global - Access global variables

SYNOPSIS

global varname ?varname ...?

DESCRIPTION

This command has no effect unless executed in the context of a proc body. If the `global` command is executed in the context of a proc body, it creates local variables linked to the corresponding global variables (though these linked variables, like those created by `upvar`, are not included in the list returned by `info locals`).

If `varname` contains namespace qualifiers, the local variable's name is the unqualified name of the global variable, as determined by the `namespace tail` command.

`varname` is always treated as the name of a variable, not an array element. An error is returned if the name looks like an array element, such as `a(b)`.

EXAMPLES

This procedure sets the namespace variable `::a::x`

```tcl
proc reset {} {
    global a::x
    set x 0
}
```
This procedure accumulates the strings passed to it in a global buffer, separated by newlines. It is useful for situations when you want to build a message piece-by-piece (as if with `puts`) but send that full message in a single piece (e.g. over a connection opened with `socket` or as part of a counted HTTP response).

```tcl
proc accum {string} {  
    global accumulator  
    append accumulator $string \n  }
```

SEE ALSO

`namespace`, `upvar`, `variable`

KEYWORDS

`global`, `namespace`, `procedure`, `variable`
NAME
msgcat - Tcl message catalog

SYNOPSIS
package require Tcl 8.5
package require msgcat 1.4.2
::msgcat::mc src-string ?arg arg ...?
::msgcat::mcmax ?src-string src-string ...?
::msgcat::mclocale ?newLocale?
::msgcat::mcpreferences
::msgcat::mcload dirname
::msgcat::mcset locale src-string ?translate-string?
::msgcat::mcmset locale src-trans-list
::msgcat::mcunknown locale src-string

LOCALE SPECIFICATION
NAMESPACEs AND MESSAGE CATALOGS
LOCATION AND FORMAT OF MESSAGE FILES
RECOMMENDED MESSAGE SETUP FOR PACKAGES
POSITIONAL CODES FOR FORMAT AND SCAN COMMANDS
CREDITS
SEE ALSO
KEYWORDS
The `msgcat` package provides a set of functions that can be used to manage multi-lingual user interfaces. Text strings are defined in a “message catalog” which is independent from the application, and which can be edited or localized without modifying the application source code. New languages or locales are provided by adding a new file to the message catalog.

Use of the message catalog is optional by any application or package, but is encouraged if the application or package wishes to be enabled for multi-lingual applications.

### Commands

::msgcat::mc `src-string` ?`arg arg ...`?

Returns a translation of `src-string` according to the user’s current locale. If additional arguments past `src-string` are given, the `format` command is used to substitute the additional arguments in the translation of `src-string`.

::msgcat::mc will search the messages defined in the current namespace for a translation of `src-string`; if none is found, it will search in the parent of the current namespace, and so on until it reaches the global namespace. If no translation string exists, ::msgcat::mcunknown is called and the string returned from ::msgcat::mcunknown is returned.

::msgcat::mc is the main function used to localize an application. Instead of using an English string directly, an application can pass the English string through ::msgcat::mc and use the result. If an
application is written for a single language in this fashion, then it is easy to add support for additional languages later simply by defining new message catalog entries.

::msgcat::mcmax ?src-string src-string ...?
Given several source strings, ::msgcat::mcmax returns the length of the longest translated string. This is useful when designing localized GUIs, which may require that all buttons, for example, be a fixed width (which will be the width of the widest button).

::msgcat::mclocale ?newLocale?
This function sets the locale to newLocale. If newLocale is omitted, the current locale is returned, otherwise the current locale is set to newLocale. msgcat stores and compares the locale in a case-insensitive manner, and returns locales in lowercase. The initial locale is determined by the locale specified in the user's environment. See LOCALE SPECIFICATION below for a description of the locale string format.

::msgcat::mcpreferences
Returns an ordered list of the locales preferred by the user, based on the user's language specification. The list is ordered from most specific to least preference. The list is derived from the current locale set in msgcat by ::msgcat::mclocale, and cannot be set independently. For example, if the current locale is en_US_funky, then ::msgcat::mcpreferences returns {en_US_funky en_US en {}}.

::msgcat::mcload dirname
Searches the specified directory for files that match the language specifications returned by ::msgcat::mcpreferences (note that these are all lowercase), extended by the file extension ".msg". Each matching file is read in order, assuming a UTF-8 encoding. The file contents are then evaluated as a Tcl script. This means that Unicode characters may be present in the message file either directly in their UTF-8 encoded form, or by use of the backslash-u quoting recognized by Tcl evaluation. The number of message files which matched the specification and were loaded is returned.
::msgcat::mcset locale src-string ?translate-string?
Sets the translation for src-string to translate-string in the specified locale and the current namespace. If translate-string is not specified, src-string is used for both. The function returns translate-string.

::msgcat::mcmset locale src-trans-list
Sets the translation for multiple source strings in src-trans-list in the specified locale and the current namespace. src-trans-list must have an even number of elements and is in the form {src-string translate-string ?src-string translate-string ...?} ::msgcat::mcmset can be significantly faster than multiple invocations of ::msgcat::mcset. The function returns the number of translations set.

::msgcat::mcunknown locale src-string
This routine is called by ::msgcat::mc in the case when a translation for src-string is not defined in the current locale. The default action is to return src-string. This procedure can be redefined by the application, for example to log error messages for each unknown string. The ::msgcat::mcunknown procedure is invoked at the same stack context as the call to ::msgcat::mc. The return value of ::msgcat::mcunknown is used as the return value for the call to ::msgcat::mc.

*****

**LOCALE SPECIFICATION**

The locale is specified to msgcat by a locale string passed to ::msgcat::mclocale. The locale string consists of a language code, an optional country code, and an optional system-specific code, each separated by “_”. The country and language codes are specified in standards ISO-639 and ISO-3166. For example, the locale “en” specifies English and “en_US” specifies U.S. English.

When the msgcat package is first loaded, the locale is initialized according to the user's environment. The variables env(LC_ALL), env(LC_MESSAGES), and env(LANG) are examined in order. The first of them to have a non-empty value is used to determine the initial
locale. The value is parsed according to the XPG4 pattern

```
language[_country][.codeset][@modifier]
```

to extract its parts. The initial locale is then set by calling ::msgcat::mclocale with the argument

```
language[_country][_modifier]
```

On Windows, if none of those environment variables is set, msgcat will attempt to extract locale information from the registry. If all these attempts to discover an initial locale from the user's environment fail, msgcat defaults to an initial locale of “C”.

When a locale is specified by the user, a “best match” search is performed during string translation. For example, if a user specifies en_GB_Funky, the locales “en_GB_Funky”, “en_GB”, “en” and “” (the empty string) are searched in order until a matching translation string is found. If no translation string is available, then ::msgcat::mcunknown is called.

**NAMESPACEs AND MESSAGE CATALOGS**

Strings stored in the message catalog are stored relative to the namespace from which they were added. This allows multiple packages to use the same strings without fear of collisions with other packages. It also allows the source string to be shorter and less prone to typographical error.

For example, executing the code

```
::msgcat::mcset en hello "hello from ::"
namespace eval foo {
    ::msgcat::mcset en hello "hello from ::foo"
```
puts [:msgcat::mc hello]
namespace eval foo {puts [:msgcat::mc hello]}

will print

```
hello from ::
hello from ::foo
```

When searching for a translation of a message, the message catalog will search first the current namespace, then the parent of the current namespace, and so on until the global namespace is reached. This allows child namespaces to “inherit” messages from their parent namespace.

For example, executing (in the “en” locale) the code

```
::msgcat::mcset en m1 ":: message1"
::msgcat::mcset en m2 ":: message2"
::msgcat::mcset en m3 ":: message3"
namespace eval ::foo {
  ::msgcat::mcset en m2 "::foo message2"
  ::msgcat::mcset en m3 "::foo message3"
}
namespace eval ::foo::bar {
  ::msgcat::mcset en m3 "::foo::bar message3"
}
namespace import ::msgcat::mc
puts "[mc m1]; [mc m2]; [mc m3]"
namespace eval ::foo {puts "[mc m1]; [mc m2]; [mc m3]"
namespace eval ::foo::bar {puts "[mc m1]; [mc m2]; [mc m3]
```

will print
LOCATION AND FORMAT OF MESSAGE FILES

Message files can be located in any directory, subject to the following conditions:

[1]
All message files for a package are in the same directory.

[2]
The message file name is a msgcat locale specifier (all lowercase) followed by “.msg”. For example:

```
es.msg     — spanish
en_gb.msg   — United Kingdom English
```

*Exception:* The message file for the root locale “” is called “ROOT.msg”. This exception is made so as not to cause peculiar behavior, such as marking the message file as “hidden” on Unix file systems.

[3]
The file contains a series of calls to `mcset` and `mcmset`, setting the necessary translation strings for the language, likely enclosed in a `namespace eval` so that all source strings are tied to the namespace of the package. For example, a short `es.msg` might contain:

```
namespace eval ::mypackage {
    ::msgcat::mcset es "Free Beer!" "Cerveza Graci
```
RECOMMENDED MESSAGE SETUP FOR PACKAGES

If a package is installed into a subdirectory of the tcl_pkgPath and loaded via package require, the following procedure is recommended.

[1]
During package installation, create a subdirectory msgs under your package directory.

[2]
Copy your *.msg files into that directory.

[3]
Add the following command to your package initialization script:

```tcl
# load language files, stored in msgs subdirectory
::msgcat::mcload [file join [file dirname [info script]]] msgs
```

POSITIONAL CODES FOR FORMAT AND SCAN COMMANDS

It is possible that a message string used as an argument to format might have positionally dependent parameters that might need to be repositioned. For example, it might be syntactically desirable to rearrange the sentence structure while translating.

format "We produced %d units in location %s" $num $city
format "In location %s we produced %d units" $city $num

This can be handled by using the positional parameters:

format "We produced %1\$d units in location %2\$s" $num $city
Similarly, positional parameters can be used with `scan` to extract values from internationalized strings.

**CREDITS**

The message catalog code was developed by Mark Harrison.

**SEE ALSO**

`format`, `scan`, `namespace`, `package`

**KEYWORDS**

internationalization, i18n, localization, l10n, message, text, translation
NAME
socket - Open a TCP network connection

SYNOPSIS
DESCRIPTION
CLIENT SOCKETS
- myaddr addr
- myport port
- async
SERVER SOCKETS
- myaddr addr
CONFIGURATION OPTIONS
- error
- sockname
- peername

EXAMPLES
SEE ALSO
KEYWORDS

NAME

socket - Open a TCP network connection

SYNOPSIS

socket ?options? host port
socket -server command ?options? port

DESCRIPTION

This command opens a network socket and returns a channel identifier that may be used in future invocations of commands like read, puts and flush. At present only the TCP network protocol is supported;
future releases may include support for additional protocols. The **socket** command may be used to open either the client or server side of a connection, depending on whether the **-server** switch is specified.

Note that the default encoding for all sockets is the system encoding, as returned by **encoding system**. Most of the time, you will need to use **fconfigure** to alter this to something else, such as **utf-8** (ideal for communicating with other Tcl processes) or **iso8859-1** (useful for many network protocols, especially the older ones).

---

### CLIENT SOCKETS

If the **-server** option is not specified, then the client side of a connection is opened and the command returns a channel identifier that can be used for both reading and writing. **Port** and **host** specify a port to connect to; there must be a server accepting connections on this port. **Port** is an integer port number (or service name, where supported and understood by the host operating system) and **host** is either a domain-style name such as **www.tcl.tk** or a numerical IP address such as **127.0.0.1**. Use **localhost** to refer to the host on which the command is invoked.

The following options may also be present before **host** to specify additional information about the connection:

- **-myaddr addr**
  
  **Addr** gives the domain-style name or numerical IP address of the client-side network interface to use for the connection. This option may be useful if the client machine has multiple network interfaces. If the option is omitted then the client-side interface will be chosen by the system software.

- **-myport port**
  
  **Port** specifies an integer port number (or service name, where supported and understood by the host operating system) to use for the client's side of the connection. If this option is omitted, the client's port number will be chosen at random by the system software.
-async

The -async option will cause the client socket to be connected asynchronously. This means that the socket will be created immediately but may not yet be connected to the server, when the call to socket returns. When a gets or flush is done on the socket before the connection attempt succeeds or fails, if the socket is in blocking mode, the operation will wait until the connection is completed or fails. If the socket is in nonblocking mode and a gets or flush is done on the socket before the connection attempt succeeds or fails, the operation returns immediately and fblocked on the socket returns 1. Synchronous client sockets may be switched (after they have connected) to operating in asynchronous mode using:

```
fconfigure chan -blocking 0
```

See the fconfigure command for more details.

SERVER SOCKETS

If the -server option is specified then the new socket will be a server for the port given by port (either an integer or a service name, where supported and understood by the host operating system; if port is zero, the operating system will allocate a free port to the server socket which may be discovered by using fconfigure to read the -sockname option). Tcl will automatically accept connections to the given port. For each connection Tcl will create a new channel that may be used to communicate with the client. Tcl then invokes command with three additional arguments: the name of the new channel, the address, in network address notation, of the client's host, and the client's port number.

The following additional option may also be specified before host:

-`myaddr addr`

Addr gives the domain-style name or numerical IP address of the
server-side network interface to use for the connection. This option may be useful if the server machine has multiple network interfaces. If the option is omitted then the server socket is bound to the special address INADDR_ANY so that it can accept connections from any interface.

Server channels cannot be used for input or output; their sole use is to accept new client connections. The channels created for each incoming client connection are opened for input and output. Closing the server channel shuts down the server so that no new connections will be accepted; however, existing connections will be unaffected.

Server sockets depend on the Tcl event mechanism to find out when new connections are opened. If the application does not enter the event loop, for example by invoking the `vwait` command or calling the C procedure `Tcl_DoOneEvent`, then no connections will be accepted.

If `port` is specified as zero, the operating system will allocate an unused port for use as a server socket. The port number actually allocated may be retrieved from the created server socket using the `fconfigure` command to retrieve the `-sockname` option as described below.

**CONFIGURATION OPTIONS**

The `fconfigure` command can be used to query several readonly configuration options for socket channels:

- **-error**
  This option gets the current error status of the given socket. This is useful when you need to determine if an asynchronous connect operation succeeded. If there was an error, the error message is returned. If there was no error, an empty string is returned.

- **-sockname**
  This option returns a list of three elements, the address, the host name and the port number for the socket. If the host name cannot be computed, the second element is identical to the address, the first element of the list.
-peername

This option is not supported by server sockets. For client and accepted sockets, this option returns a list of three elements; these are the address, the host name and the port to which the peer socket is connected or bound. If the host name cannot be computed, the second element of the list is identical to the address, its first element.

**EXAMPLES**

Here is a very simple time server:

```tcl
proc Server {channel clientaddr clientport} {
    puts "Connection from $clientaddr registered"
    puts $channel [clock format [clock seconds]]
    close $channel
}

socket -server Server 9900
vwait forever
```

And here is the corresponding client to talk to the server:

```tcl
set server localhost
set sockChan [socket $server 9900]
gets $sockChan line
close $sockChan
puts "The time on $server is $line"
```

**SEE ALSO**

fconfigure, flush, open, read

**KEYWORDS**
bind, channel, connection, domain name, host, network address, socket, tcp
NAME

variable - create and initialize a namespace variable

SYNOPSIS

variable ?name value...? name ?value?

DESCRIPTION

This command is normally used within a namespace eval command to create one or more variables within a namespace. Each variable name is initialized with value. The value for the last variable is optional.

If a variable name does not exist, it is created. In this case, if value is specified, it is assigned to the newly created variable. If no value is specified, the new variable is left undefined. If the variable already exists, it is set to value if value is specified or left unchanged if no value is given. Normally, name is unqualified (does not include the names of any containing namespaces), and the variable is created in the current namespace. If name includes any namespace qualifiers, the variable is created in the specified namespace. If the variable is not defined, it will be visible to the namespace which command, but not to the info exists command.

If the variable command is executed inside a Tcl procedure, it creates local variables linked to the corresponding namespace variables (and therefore these variables are listed by info vars.) In this way the variable command resembles the global command, although the global command only links to variables in the global namespace. If any values are given, they are used to modify the values of the associated namespace variables. If a namespace variable does not exist, it is
created and optionally initialized.

A *name* argument cannot reference an element within an array. Instead, *name* should reference the entire array, and the initialization *value* should be left off. After the variable has been declared, elements within the array can be set using ordinary `set` or `array` commands.

**EXAMPLES**

Create a variable in a namespace:

```bash
namespace eval foo {
    variable bar 12345
}
```

Create an array in a namespace:

```bash
namespace eval someNS {
    variable someAry
    array set someAry {
        someName   someValue
        otherName  otherValue
    }
}
```

Access variables in namespaces from a procedure:

```bash
namespace eval foo {
    proc spong {} {
        # Variable in this namespace
        variable bar
        puts "bar is $bar"

        # Variable in another namespace
```
variable ::someNS::someAry
parray someAry
}
}

SEE ALSO

global, namespace, upvar

KEYWORDS

global, namespace, procedure, variable

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright © 1997 Sun Microsystems, Inc.
NAME

close - Close an open channel

SYNOPSIS

close channelId

DESCRIPTION

Closes the channel given by channelId.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

All buffered output is flushed to the channel's output device, any buffered input is discarded, the underlying file or device is closed, and channelId becomes unavailable for use.

If the channel is blocking, the command does not return until all output is flushed. If the channel is nonblocking and there is unflushed output, the channel remains open and the command returns immediately; output will be flushed in the background and the channel will be closed when all the flushing is complete.

If channelId is a blocking channel for a command pipeline then close waits for the child processes to complete.

If the channel is shared between interpreters, then close makes channelId unavailable in the invoking interpreter but has no other effect until all of the sharing interpreters have closed the channel. When the
last interpreter in which the channel is registered invokes close, the cleanup actions described above occur. See the interp command for a description of channel sharing.

Channels are automatically closed when an interpreter is destroyed and when the process exits. Channels are switched to blocking mode, to ensure that all output is correctly flushed before the process exits.

The command returns an empty string, and may generate an error if an error occurs while flushing output. If a command in a command pipeline created with open returns an error, close generates an error (similar to the exec command.)

**EXAMPLE**

This illustrates how you can use Tcl to ensure that files get closed even when errors happen by combining catch, close and return:

```tcl
proc withOpenFile {filename channelVar script} {
    upvar 1 $channelVar chan
    set chan [open $filename]
    catch {
        uplevel 1 $script
    } result options
    close $chan
    return -options $options $result
}
```

**SEE ALSO**

file, open, socket, eof, Tcl_StandardChannels

**KEYWORDS**

blocking, channel, close, nonblocking
**NAME**

history - Manipulate the history list

**SYNOPSIS**

**DESCRIPTION**

history
history add command ?exec?
history change newValue ?event?
history clear
history event ?event?
history info ?count?
history keep ?count?
history nextid
history redo ?event?

The `history` command performs one of several operations related to recently-executed commands recorded in a history list. Each of these recorded commands is referred to as an “event”. When specifying an event to the `history` command, the following forms may be used:

[1]
A number: if positive, it refers to the event with that number (all events are numbered starting at 1). If the number is negative, it selects an event relative to the current event (-1 refers to the previous event, -2 to the one before that, and so on). Event 0 refers to the current event.

[2] A string: selects the most recent event that matches the string. An event is considered to match the string either if the string is the same as the first characters of the event, or if the string matches the event in the sense of the string match command.

The history command can take any of the following forms:

history
   Same as history info, described below.

history add command ?exec?
   Adds the command argument to the history list as a new event. If exec is specified (or abbreviated) then the command is also executed and its result is returned. If exec is not specified then an empty string is returned as result.

history change newValue ?event?
   Replaces the value recorded for an event with newValue. Event specifies the event to replace, and defaults to the current event (not event -1). This command is intended for use in commands that implement new forms of history substitution and wish to replace the current event (which invokes the substitution) with the command created through substitution. The return value is an empty string.

history clear
   Erase the history list. The current keep limit is retained. The history event numbers are reset.

history event ?event?
   Returns the value of the event given by event. Event defaults to -1.
history info ?count?
Returns a formatted string (intended for humans to read) giving the event number and contents for each of the events in the history list except the current event. If count is specified then only the most recent count events are returned.

history keep ?count?
This command may be used to change the size of the history list to count events. Initially, 20 events are retained in the history list. If count is not specified, the current keep limit is returned.

history nextid
Returns the number of the next event to be recorded in the history list. It is useful for things like printing the event number in command-line prompts.

history redo ?event?
Re-executes the command indicated by event and returns its result. Event defaults to -1. This command results in history revision: see below for details.

HISTORY REVISION

Pre-8.0 Tcl had a complex history revision mechanism. The current mechanism is more limited, and the old history operations substitute and words have been removed. (As a consolation, the clear operation was added.)

The history option redo results in much simpler “history revision”. When this option is invoked then the most recent event is modified to eliminate the history command and replace it with the result of the history command. If you want to redo an event without modifying history, then use the event operation to retrieve some event, and the add operation to add it to history and execute it.

KEYWORDS

event, history, record
NAME

namespace - create and manipulate contexts for commands and variables

SYNOPSIS

DESCRIPTION

namespace children \?namespace\? \?pattern\?
namespace code script
namespace current
namespace delete \?namespace namespace \?...
namespace ensemble subcommand \?arg \?...
namespace eval namespace arg \?arg \?...
namespace exists namespace
namespace export \?-clear\? \?pattern pattern \?...
namespace forget \?pattern pattern \?...
namespace import \?-force\? \?pattern pattern \?...
namespace inscope namespace script \?arg \?...
namespace origin command
namespace parent \?namespace?
namespace path \?namespaceList?
namespace qualifiers string
namespace tail string
namespace upvar namespace otherVar myVar \?otherVar

myVar ...
namespace unknown \?script?
namespace which \?-command\? \?-variable\? name

WHAT IS A NAMESPACE?
QUALIFIED NAMES
NAME RESOLUTION
IMPORTING COMMANDS
EXPORTING COMMANDS
SCOPED SCRIPTS
namespace - create and manipulate contexts for commands and variables

namespace children namespace? ?pattern?
Returns a list of all child namespaces that belong to the namespace namespace. If namespace is not specified, then the children are returned for the current namespace. This command
returns fully-qualified names, which start with a double colon (::). If the optional pattern is given, then this command returns only the names that match the glob-style pattern. The actual pattern used is determined as follows: a pattern that starts with double colon (::) is used directly, otherwise the namespace namespace (or the fully-qualified name of the current namespace) is prepended onto the pattern.

**namespace code script**
Captures the current namespace context for later execution of the script script. It returns a new script in which script has been wrapped in a namespace inscope command. The new script has two important properties. First, it can be evaluated in any namespace and will cause script to be evaluated in the current namespace (the one where the namespace code command was invoked). Second, additional arguments can be appended to the resulting script and they will be passed to script as additional arguments. For example, suppose the command set script [namespace code {foo bar}] is invoked in namespace ::a::b. Then eval $script [list x y] can be executed in any namespace (assuming the value of script has been passed in properly) and will have the same effect as the command ::namespace eval ::a::b {foo bar x y}. This command is needed because extensions like Tk normally execute callback scripts in the global namespace. A scoped command captures a command together with its namespace context in a way that allows it to be executed properly later. See the section SCOPED SCRIPTS for some examples of how this is used to create callback scripts.

**namespace current**
Returns the fully-qualified name for the current namespace. The actual name of the global namespace is "" (i.e., an empty string), but this command returns :: for the global namespace as a convenience to programmers.

**namespace delete ?namespace namespace ...?**
Each namespace namespace is deleted and all variables, procedures, and child namespaces contained in the namespace
are deleted. If a procedure is currently executing inside the namespace, the namespace will be kept alive until the procedure returns; however, the namespace is marked to prevent other code from looking it up by name. If a namespace does not exist, this command returns an error. If no namespace names are given, this command does nothing.

**namespace ensemble subcommand ?arg ...?**

Creates and manipulates a command that is formed out of an ensemble of subcommands. See the section **ENSEMBLES** below for further details.

**namespace eval namespace arg ?arg ...?**

Activates a namespace called `namespace` and evaluates some code in that context. If the namespace does not already exist, it is created. If more than one `arg` argument is specified, the arguments are concatenated together with a space between each one in the same fashion as the `eval` command, and the result is evaluated.

If `namespace` has leading namespace qualifiers and any leading namespaces do not exist, they are automatically created.

**namespace exists namespace**

Returns **1** if `namespace` is a valid namespace in the current context, returns **0** otherwise.

**namespace export ?-clear? ?pattern pattern ...?**

Specifies which commands are exported from a namespace. The exported commands are those that can be later imported into another namespace using a `namespace import` command. Both commands defined in a namespace and commands the namespace has previously imported can be exported by a namespace. The commands do not have to be defined at the time the `namespace export` command is executed. Each `pattern` may contain glob-style special characters, but it may not include any namespace qualifiers. That is, the pattern can only specify commands in the current (exporting) namespace. Each `pattern` is appended onto the namespace's list of export patterns. If the `-clear`
flag is given, the namespace's export pattern list is reset to empty before any pattern arguments are appended. If no patterns are given and the -clear flag is not given, this command returns the namespace's current export list.

**namespace forget ?pattern pattern ...**?

Removes previously imported commands from a namespace. Each pattern is a simple or qualified name such as x, foo::x or a::b::p*. Qualified names contain double colons (::) and qualify a name with the name of one or more namespaces. Each “qualified pattern” is qualified with the name of an exporting namespace and may have glob-style special characters in the command name at the end of the qualified name. Glob characters may not appear in a namespace name. For each “simple pattern” this command deletes the matching commands of the current namespace that were imported from a different namespace. For “qualified patterns”, this command first finds the matching exported commands. It then checks whether any of those commands were previously imported by the current namespace. If so, this command deletes the corresponding imported commands. In effect, this un-does the action of a namespace import command.

**namespace import ?-force? ?pattern pattern ...**?

Imports commands into a namespace, or queries the set of imported commands in a namespace. When no arguments are present, namespace import returns the list of commands in the current namespace that have been imported from other namespaces. The commands in the returned list are in the format of simple names, with no namespace qualifiers at all. This format is suitable for composition with namespace forget (see EXAMPLES below). When pattern arguments are present, each pattern is a qualified name like foo::x or a::p*. That is, it includes the name of an exporting namespace and may have glob-style special characters in the command name at the end of the qualified name. Glob characters may not appear in a namespace name. All the commands that match a pattern string and which are currently exported from their namespace are added to the current
namespace. This is done by creating a new command in the current namespace that points to the exported command in its original namespace; when the new imported command is called, it invokes the exported command. This command normally returns an error if an imported command conflicts with an existing command. However, if the -force option is given, imported commands will silently replace existing commands. The namespace import command has snapshot semantics: that is, only requested commands that are currently defined in the exporting namespace are imported. In other words, you can import only the commands that are in a namespace at the time when the namespace import command is executed. If another command is defined and exported in this namespace later on, it will not be imported.

namespace inscope namespace script ?arg ...?
Executes a script in the context of the specified namespace. This command is not expected to be used directly by programmers; calls to it are generated implicitly when applications use namespace code commands to create callback scripts that the applications then register with, e.g., Tk widgets. The namespace inscope command is much like the namespace eval command except that the namespace must already exist, and namespace inscope appends additional args as proper list elements.

\begin{verbatim}
namespace inscope ::foo $script $x $y $z
\end{verbatim}

is equivalent to

\begin{verbatim}
namespace eval ::foo [concat $script [list $x $y
\end{verbatim}

thus additional arguments will not undergo a second round of substitution, as is the case with namespace eval.

namespace origin command
Returns the fully-qualified name of the original command to which the imported command `command` refers. When a command is imported into a namespace, a new command is created in that namespace that points to the actual command in the exporting namespace. If a command is imported into a sequence of namespaces `a, b,...,n` where each successive namespace just imports the command from the previous namespace, this command returns the fully-qualified name of the original command in the first namespace, `a`. If `command` does not refer to an imported command, the command's own fully-qualified name is returned.

`namespace parent ?namespace?`  
Returns the fully-qualified name of the parent namespace for namespace `namespace`. If `namespace` is not specified, the fully-qualified name of the current namespace's parent is returned.

`namespace path ?namespaceList?`  
Returns the command resolution path of the current namespace. If `namespaceList` is specified as a list of named namespaces, the current namespace's command resolution path is set to those namespaces and returns the empty list. The default command resolution path is always empty. See the section **NAME RESOLUTION** below for an explanation of the rules regarding name resolution.

`namespace qualifiers string`  
Returns any leading namespace qualifiers for `string`. Qualifiers are namespace names separated by double colons (::). For the `string ::foo::bar::x`, this command returns `::foo::bar`, and for :: it returns an empty string. This command is the complement of the `namespace tail` command. Note that it does not check whether the namespace names are, in fact, the names of currently defined namespaces.

`namespace tail string`  
Returns the simple name at the end of a qualified string. Qualifiers are namespace names separated by double colons (::). For the `string ::foo::bar::x`, this command returns `x`, and for :: it returns an
empty string. This command is the complement of the namespace qualifiers command. It does not check whether the namespace names are, in fact, the names of currently defined namespaces.

namespace upvar namespace otherVar myVar ?otherVar myVar ...
This command arranges for one or more local variables in the current procedure to refer to variables in namespace. The namespace name is resolved as described in section NAME RESOLUTION. The command namespace upvar $ns a b has the same behaviour as upvar 0 ${ns}::a b, with the sole exception of the resolution rules used for qualified namespace or variable names. namespace upvar returns an empty string.

namespace unknown ?script?
Sets or returns the unknown command handler for the current namespace. The handler is invoked when a command called from within the namespace cannot be found (in either the current namespace or the global namespace). The script argument, if given, should be a well formed list representing a command name and optional arguments. When the handler is invoked, the full invocation line will be appended to the script and the result evaluated in the context of the namespace. The default handler for all namespaces is ::unknown. If no argument is given, it returns the handler for the current namespace.

namespace which ?-command? ?-variable? name
Looks up name as either a command or variable and returns its fully-qualified name. For example, if name does not exist in the current namespace but does exist in the global namespace, this command returns a fully-qualified name in the global namespace. If the command or variable does not exist, this command returns an empty string. If the variable has been created but not defined, such as with the variable command or through a trace on the variable, this command will return the fully-qualified name of the variable. If no flag is given, name is treated as a command name. See the section NAME RESOLUTION below for an explanation of the rules regarding name resolution.
WHAT IS A NAMESPACE?

A namespace is a collection of commands and variables. It encapsulates the commands and variables to ensure that they will not interfere with the commands and variables of other namespaces. Tcl has always had one such collection, which we refer to as the *global namespace*. The global namespace holds all global variables and commands. The `namespace eval` command lets you create new namespaces. For example,

```tcl
namespace eval Counter {
    namespace export bump
    variable num 0

    proc bump {} {
        variable num
        incr num
    }
}
```

creates a new namespace containing the variable `num` and the procedure `bump`. The commands and variables in this namespace are separate from other commands and variables in the same program. If there is a command named `bump` in the global namespace, for example, it will be different from the command `bump` in the `Counter` namespace.

Namespace variables resemble global variables in Tcl. They exist outside of the procedures in a namespace but can be accessed in a procedure via the `variable` command, as shown in the example above.

Namespaces are dynamic. You can add and delete commands and variables at any time, so you can build up the contents of a namespace over time using a series of `namespace eval` commands. For example, the following series of commands has the same effect as the namespace definition shown above:
namespace eval Counter {
    variable num 0
    proc bump {} {
        variable num
        return [incr num]
    }
}
namespace eval Counter {
    proc test {args} {
        return $args
    }
}
namespace eval Counter {
    rename test ""
}

Note that the test procedure is added to the Counter namespace, and later removed via the rename command.

Namespaces can have other namespaces within them, so they nest hierarchically. A nested namespace is encapsulated inside its parent namespace and can not interfere with other namespaces.

QUALIFIED NAMES

Each namespace has a textual name such as history or ::safe::interp. Since namespaces may nest, qualified names are used to refer to commands, variables, and child namespaces contained inside namespaces. Qualified names are similar to the hierarchical path names for Unix files or Tk widgets, except that :: is used as the separator instead of / or .. The topmost or global namespace has the name "" (i.e., an empty string), although :: is a synonym. As an example, the name ::safe::interp::create refers to the command create in the namespace interp that is a child of namespace ::safe, which in turn is a child of the global namespace, ::.
If you want to access commands and variables from another namespace, you must use some extra syntax. Names must be qualified by the namespace that contains them. From the global namespace, we might access the **Counter** procedures like this:

```
Counter::bump 5
Counter::Reset
```

We could access the current count like this:

```
puts "count = $Counter::num"
```

When one namespace contains another, you may need more than one qualifier to reach its elements. If we had a namespace **Foo** that contained the namespace **Counter**, you could invoke its **bump** procedure from the global namespace like this:

```
Foo::Counter::bump 3
```

You can also use qualified names when you create and rename commands. For example, you could add a procedure to the **Foo** namespace like this:

```
proc Foo::Test {args} {return $args}
```

And you could move the same procedure to another namespace like this:

```
rename Foo::Test Bar::Test
```
There are a few remaining points about qualified names that we should cover. Namespaces have nonempty names except for the global namespace. :: is disallowed in simple command, variable, and namespace names except as a namespace separator. Extra colons in any separator part of a qualified name are ignored; i.e. two or more colons are treated as a namespace separator. A trailing :: in a qualified variable or command name refers to the variable or command named {}. However, a trailing :: in a qualified namespace name is ignored.

**NAME RESOLUTION**

In general, all Tcl commands that take variable and command names support qualified names. This means you can give qualified names to such commands as `set`, `proc`, `rename`, and `interp alias`. If you provide a fully-qualified name that starts with a ::, there is no question about what command, variable, or namespace you mean. However, if the name does not start with a :: (i.e., is relative), Tcl follows basic rules for looking it up: Variable names are always resolved by looking first in the current namespace, and then in the global namespace. Command names are also always resolved by looking in the current namespace first. If not found there, they are searched for in every namespace on the current namespace's command path (which is empty by default). If not found there, command names are looked up in the global namespace (or, failing that, are processed by the `unknown` command.) Namespace names, on the other hand, are always resolved by looking in only the current namespace.

In the following example,

```tcl
set traceLevel 0
namespace eval Debug {
    printTrace $traceLevel
}
```

Tcl looks for `traceLevel` in the namespace `Debug` and then in the global namespace. It looks up the command `printTrace` in the same
way. If a variable or command name is not found in either context, the name is undefined. To make this point absolutely clear, consider the following example:

```tcl
set traceLevel 0
namespace eval Foo {
    variable traceLevel 3

    namespace eval Debug {
        printTrace $traceLevel
    }
}
```

Here Tcl looks for `traceLevel` first in the namespace `Foo::Debug`. Since it is not found there, Tcl then looks for it in the global namespace. The variable `Foo::traceLevel` is completely ignored during the name resolution process.

You can use the `namespace which` command to clear up any question about name resolution. For example, the command:

```tcl
namespace eval Foo::Debug {namespace which -variable traceLevel}
```

returns `::traceLevel`. On the other hand, the command,

```tcl
namespace eval Foo {namespace which -variable traceLevel}
```

returns `::Foo::traceLevel`.

As mentioned above, namespace names are looked up differently than the names of variables and commands. Namespace names are always resolved in the current namespace. This means, for example, that a
The `namespace eval` command that creates a new namespace always creates a child of the current namespace unless the new namespace name begins with `::`.

Tcl has no access control to limit what variables, commands, or namespaces you can reference. If you provide a qualified name that resolves to an element by the name resolution rule above, you can access the element.

You can access a namespace variable from a procedure in the same namespace by using the `variable` command. Much like the `global` command, this creates a local link to the namespace variable. If necessary, it also creates the variable in the current namespace and initializes it. Note that the `global` command only creates links to variables in the global namespace. It is not necessary to use a `variable` command if you always refer to the namespace variable using an appropriate qualified name.

## Importing Commands

Namespaces are often used to represent libraries. Some library commands are used so frequently that it is a nuisance to type their qualified names. For example, suppose that all of the commands in a package like BLT are contained in a namespace called `Blt`. Then you might access these commands like this:

```
Blt::graph .g -background red
Blt::table .g 0,0
```

If you use the `graph` and `table` commands frequently, you may want to access them without the `Blt::` prefix. You can do this by importing the commands into the current namespace, like this:

```
namespace import Blt::*
```
This adds all exported commands from the BLt namespace into the current namespace context, so you can write code like this:

```
graph .g -background red
table .g 0,0
```

The `namespace import` command only imports commands from a namespace that that namespace exported with a `namespace export` command.

Importing every command from a namespace is generally a bad idea since you do not know what you will get. It is better to import just the specific commands you need. For example, the command

```
namespace import Blt::graph Blt::table
```

imports only the `graph` and `table` commands into the current context.

If you try to import a command that already exists, you will get an error. This prevents you from importing the same command from two different packages. But from time to time (perhaps when debugging), you may want to get around this restriction. You may want to reissue the `namespace import` command to pick up new commands that have appeared in a namespace. In that case, you can use the `-force` option, and existing commands will be silently overwritten:

```
namespace import -force Blt::graph Blt::table
```

If for some reason, you want to stop using the imported commands, you can remove them with a `namespace forget` command, like this:

```
namespace forget Blt::*
```
This searches the current namespace for any commands imported from Blt. If it finds any, it removes them. Otherwise, it does nothing. After this, the Blt commands must be accessed with the Blt:: prefix.

When you delete a command from the exporting namespace like this:

```
rename Blt::graph ""
```

the command is automatically removed from all namespaces that import it.

## EXPORTING COMMANDS

You can export commands from a namespace like this:

```bash
namespace eval Counter {
    namespace export bump reset
    variable Num 0
    variable Max 100

    proc bump {{by 1}} {
        variable Num
        incr Num $by
        Check
        return $Num
    }

    proc reset {} {
        variable Num
        set Num 0
    }

    proc Check {} {
        variable Num
        variable Max
        if {$Num > $Max} {
```
The procedures **bump** and **reset** are exported, so they are included when you import from the **Counter** namespace, like this:

```
namespace import Counter::*
```

However, the **Check** procedure is not exported, so it is ignored by the import operation.

The **namespace import** command only imports commands that were declared as exported by their namespace. The **namespace export** command specifies what commands may be imported by other namespaces. If a **namespace import** command specifies a command that is not exported, the command is not imported.

### SCOPED SCRIPTS

The **namespace code** command is the means by which a script may be packaged for evaluation in a namespace other than the one in which it was created. It is used most often to create event handlers, Tk bindings, and traces for evaluation in the global context. For instance, the following code indicates how to direct a variable trace callback into the current namespace:

```
namespace eval a {
    variable b
    proc theTraceCallback { n1 n2 op } {
        upvar 1 $n1 var
        puts "the value of $n1 has changed to $var"
        return
    }
```
When executed, it prints the message:

```
the value of a::b has changed to c
```

**ENSEMBLES**

The `namespace ensemble` is used to create and manipulate ensemble commands, which are commands formed by grouping subcommands together. The commands typically come from the current namespace when the ensemble was created, though this is configurable. Note that there may be any number of ensembles associated with any namespace (including none, which is true of all namespaces by default), though all the ensembles associated with a namespace are deleted when that namespace is deleted. The link between an ensemble command and its namespace is maintained however the ensemble is renamed.

Three subcommands of the `namespace ensemble` command are defined:

```
namespace ensemble create ?option value ...?
```

Creates a new ensemble command linked to the current namespace, returning the fully qualified name of the command created. The arguments to `namespace ensemble create` allow the configuration of the command as if with the `namespace ensemble configure` command. If not overridden with the `-command` option, this command creates an ensemble with exactly the same name as the linked namespace. See the section **ENSEMBLE OPTIONS** below for a full list of options supported and their effects.

```
namespace ensemble configure command ?option? ?value ...?
```
Retrieves the value of an option associated with the ensemble command named \textit{command}, or updates some options associated with that ensemble command. See the section \textbf{ENSEMBLE OPTIONS} below for a full list of options supported and their effects.

\textbf{namespace ensemble exists \textit{command}}

Returns a boolean value that describes whether the command \textit{command} exists and is an ensemble command. This command only ever returns an error if the number of arguments to the command is wrong.

When called, an ensemble command takes its first argument and looks it up (according to the rules described below) to discover a list of words to replace the ensemble command and subcommand with. The resulting list of words is then evaluated (with no further substitutions) as if that was what was typed originally (i.e. by passing the list of words through \texttt{Tcl\_EvalObjv}) and returning the result of the command. Note that it is legal to make the target of an ensemble rewrite be another (or even the same) ensemble command. The ensemble command will not be visible through the use of the \texttt{uplevel} or \texttt{info level} commands.

\textbf{ENSEMBLE OPTIONS}

The following options, supported by the \texttt{namespace ensemble create} and \texttt{namespace ensemble configure} commands, control how an ensemble command behaves:

\textbf{-map}

When non-empty, this option supplies a dictionary that provides a mapping from subcommand names to a list of prefix words to substitute in place of the ensemble command and subcommand words (in a manner similar to an alias created with \texttt{interp alias}; the words are not reparsed after substitution). When this option is empty, the mapping will be from the local name of the subcommand to its fully-qualified name. Note that when this option is non-empty and the \texttt{-subcommands} option is empty, the ensemble subcommand names will be exactly those words that have
mappings in the dictionary.

-prefixes
This option (which is enabled by default) controls whether the
ensemble command recognizes unambiguous prefixes of its
subcommands. When turned off, the ensemble command requires
exact matching of subcommand names.

-subcommands
When non-empty, this option lists exactly what subcommands are
in the ensemble. The mapping for each of those commands will be
either whatever is defined in the -map option, or to the command
with the same name in the namespace linked to the ensemble. If
this option is empty, the subcommands of the namespace will either
be the keys of the dictionary listed in the -map option or the
exported commands of the linked namespace at the time of the
invocation of the ensemble command.

-unknown
When non-empty, this option provides a partial command (to which
all the words that are arguments to the ensemble command,
including the fully-qualified name of the ensemble, are appended)
to handle the case where an ensemble subcommand is not
recognized and would otherwise generate an error. When empty
(the default) an error (in the style of Tcl_GetIndexFromObj) is
generated whenever the ensemble is unable to determine how to
implement a particular subcommand. See UNKNOWN HANDLER
BEHAVIOUR for more details.

The following extra option is allowed by namespace ensemble create:

-command
This write-only option allows the name of the ensemble created by
namespace ensemble create to be anything in any existing
namespace. The default value for this option is the fully-qualified
name of the namespace in which the namespace ensemble
create command is invoked.
The following extra option is allowed by `namespace ensemble configure`:

**-namespace**
This read-only option allows the retrieval of the fully-qualified name of the namespace which the ensemble was created within.

### UNKNOWN HANDLER BEHAVIOUR

If an unknown handler is specified for an ensemble, that handler is called when the ensemble command would otherwise return an error due to it being unable to decide which subcommand to invoke. The exact conditions under which that occurs are controlled by the `-subcommands`, `-map` and `-prefixes` options as described above.

To execute the unknown handler, the ensemble mechanism takes the specified `-unknown` option and appends each argument of the attempted ensemble command invocation (including the ensemble command itself, expressed as a fully qualified name). It invokes the resulting command in the scope of the attempted call. If the execution of the unknown handler terminates normally, the ensemble engine reparses the subcommand (as described below) and tries to dispatch it again, which is ideal for when the ensemble's configuration has been updated by the unknown subcommand handler. Any other kind of termination of the unknown handler is treated as an error.

The result of the unknown handler is expected to be a list (it is an error if it is not). If the list is an empty list, the ensemble command attempts to look up the original subcommand again and, if it is not found this time, an error will be generated just as if the `-unknown` handler was not there (i.e. for any particular invocation of an ensemble, its unknown handler will be called at most once.) This makes it easy for the unknown handler to update the ensemble or its backing namespace so as to provide an implementation of the desired subcommand and reparse.

When the result is a non-empty list, the words of that list are used to replace the ensemble command and subcommand, just as if they had been looked up in the `-map`. It is up to the unknown handler to supply
all namespace qualifiers if the implementing subcommand is not in the namespace of the caller of the ensemble command. Also note that when ensemble commands are chained (e.g. if you make one of the commands that implement an ensemble subcommand into an ensemble, in a manner similar to the text widget's tag and mark subcommands) then the rewrite happens in the context of the caller of the outermost ensemble. That is to say that ensembles do not in themselves place any namespace contexts on the Tcl call stack.

Where an empty -unknown handler is given (the default), the ensemble command will generate an error message based on the list of commands that the ensemble has defined (formatted similarly to the error message from Tcl_GetIndexFromObj). This is the error that will be thrown when the subcommand is still not recognized during reparsing. It is also an error for an -unknown handler to delete its namespace.

EXAMPLES

Create a namespace containing a variable and an exported command:

```tcl
namespace eval foo { 
    variable bar 0 
    proc grill {} { 
        variable bar 
        puts "called [incr bar] times"
    } 
    namespace export grill
}
```

Call the command defined in the previous example in various ways.

```tcl
# Direct call
::foo::grill
```
# Use the command resolution path to find the name
namespace eval boo {
    namespace path ::foo
    grill
}

# Import into current namespace, then call local alias
namespace import foo::grill
grill

# Create two ensembles, one with the default name and one with a specified name. Then call through the ensembles.
namespace eval foo {
    namespace ensemble create
    namespace ensemble create -command ::foobar
}
foo grill
foobar grill

Look up where the command imported in the previous example came from:

puts "grill came from [namespace origin grill]"

Remove all imported commands from the current namespace:

namespace forget {*}[namespace import]

SEE ALSO

interp, upvar, variable

KEYWORDS
command, ensemble, exported, internal, variable
source - Evaluate a file or resource as a Tcl script

**SYNOPSIS**

```source fileName```
```source -encoding encodingName fileName```

**DESCRIPTION**

This command takes the contents of the specified file or resource and passes it to the Tcl interpreter as a text script. The return value from `source` is the return value of the last command executed in the script. If an error occurs in evaluating the contents of the script then the `source` command will return that error. If a `return` command is invoked from within the script then the remainder of the file will be skipped and the `source` command will return normally with the result from the `return` command.

The end-of-file character for files is “\32” (^Z) for all platforms. The `source` command will read files up to this character. This restriction does not exist for the `read` or `gets` commands, allowing for files containing code and data segments (scripted documents). If you require a “^Z” in code for string comparison, you can use “\032” or “\u001a”, which will be safely substituted by the Tcl interpreter into “^Z”.

The `-encoding` option is used to specify the encoding of the data stored in `fileName`. When the `-encoding` option is omitted, the system encoding is assumed.

**EXAMPLE**
Run the script in the file **foo.tcl** and then the script in the file **bar.tcl**:

```
source foo.tcl
source bar.tcl
```

Alternatively:

```
foreach scriptFile {foo.tcl bar.tcl} {
    source $scriptFile
}
```

**SEE ALSO**

[file, cd, encoding, info]

**KEYWORDS**

[file, script]
NAME

vwait - Process events until a variable is written

SYNOPSIS

vwait varName

DESCRIPTION

This command enters the Tcl event loop to process events, blocking the application if no events are ready. It continues processing events until some event handler sets the value of variable varName. Once varName has been set, the vwait command will return as soon as the event handler that modified varName completes. varName must be globally scoped (either with a call to global for the varName, or with the full namespace path specification).

In some cases the vwait command may not return immediately after varName is set. This can happen if the event handler that sets varName does not complete immediately. For example, if an event handler sets varName and then itself calls vwait to wait for a different variable, then it may not return for a long time. During this time the top-level vwait is blocked waiting for the event handler to complete, so it cannot return either.

EXAMPLES

Run the event-loop continually until some event calls exit. (You can use any variable not mentioned elsewhere, but the name forever reminds you at a glance of the intent.)
vwait forever

Wait five seconds for a connection to a server socket, otherwise close the socket and continue running the script:

```
# Initialise the state
after 5000 set state timeout
set server [socket -server accept 12345]
proc accept {args} {
    global state connectionInfo
    set state accepted
    set connectionInfo $args
}

# Wait for something to happen
vwait state

# Clean up events that could have happened
close $server
after cancel set state timeout

# Do something based on how the vwait finished...
switch $state {
    timeout {
        puts "no connection on port 12345"
    }
    accepted {
        puts "connection: $connectionInfo"
        puts [lindex $connectionInfo 0] "Hello there!"
    }
}
```

SEE ALSO
NAME

concat - Join lists together

SYNOPSIS

concat ?arg arg ...?

DESCRIPTION

This command joins each of its arguments together with spaces after trimming leading and trailing white-space from each of them. If all the arguments are lists, this has the same effect as concatenating them into a single list. It permits any number of arguments; if no args are supplied, the result is an empty string.

EXAMPLES

Although concat will concatenate lists, flattening them in the process (so giving the following interactive session):

```
% concat a b {c d e} {f {g h}}
```

```
a b c d e f {g h}
```

it will also concatenate things that are not lists, as can be seen from this session:

```
% concat " a b {c " d " e} f"
```

```
a b {c d e} f
```
Note also that the concatenation does not remove spaces from the middle of values, as can be seen here:

```%
concat "a  b  c" { d e f }
```

(a,b,c d e f)

(i.e., there are three spaces between each of the a, the b and the c).

**SEE ALSO**

append, eval, join

**KEYWORDS**

concatenate, join, lists
NAME
http - Client-side implementation of the HTTP/1.1 protocol

SYNOPSIS

DESCRIPTION

COMMANDS

::http::config ?options?
    -accept mimetypes
    -proxyhost hostname
    -proxyport number
    -proxyfilter command
    -urlencode encoding
    -useragent string

::http::geturl url ?options?
    -binary boolean
    -blocksize size
    -channel name
    -command callback
    -handler callback
    -headers keyvaluelist
    -keepalive boolean
    -method type
    -myaddr address
    -progress callback
    -protocol version
    -query query
    -queryblocksize size
    -querychannel channelID
    -queryprogress callback
    -strict boolean
    -timeout milliseconds
    -type mime-type
-validate boolean
::http::formatQuery key value ?key value ...?
::http::reset token ?why?
::http::wait token
::http::data token
::http::error token
::http::status token
::http::code token
::http::ncode token
::http::size token
::http::meta token
::http::cleanup token
::http::register proto port command
::http::unregister proto

ERRORS
ok
eof
error

STATE ARRAY
body
charset
coding
currentsize
error
http
meta

      Content-Type
      Content-Length
      Location
posterror
status
totalsize
type
url

EXAMPLE
SEE ALSO
KEYWORDS
NAME

http - Client-side implementation of the HTTP/1.1 protocol

SYNOPSIS

package require http ?2.7?
::http::config ?options?
::http::geturl url ?options?
::http::formatQuery key value ?key value ...?
::http::reset token ?why?
::http::wait token
::http::status token
::http::size token
::http::code token
::http::ncode token
::http::meta token
::http::data token
::http::error token
::http::cleanup token
::http::register proto port command
::http::unregister proto

DESCRIPTION

The http package provides the client side of the HTTP/1.1 protocol. The package implements the GET, POST, and HEAD operations of HTTP/1.1. It allows configuration of a proxy host to get through firewalls. The package is compatible with the Safesock security policy, so it can be used by untrusted applets to do URL fetching from a restricted set of hosts. This package can be extended to support additional HTTP transport protocols, such as HTTPS, by providing a custom socket command, via ::http::register.

The ::http::geturl procedure does a HTTP transaction. Its options determine whether a GET, POST, or HEAD transaction is performed. The return value of ::http::geturl is a token for the transaction. The value is also the name of an array in the ::http namespace that contains
state information about the transaction. The elements of this array are
described in the **STATE ARRAY** section.

If the `-command` option is specified, then the HTTP operation is done in
the background. `::http::geturl` returns immediately after generating the
HTTP request and the callback is invoked when the transaction
completes. For this to work, the Tcl event loop must be active. In Tk
applications this is always true. For pure-Tcl applications, the caller can
use `::http::wait` after calling `::http::geturl` to start the event loop.

### COMMANDS

**::http::config ?options?**
The `::http::config` command is used to set and query the name of
the proxy server and port, and the User-Agent name used in the
HTTP requests. If no options are specified, then the current
configuration is returned. If a single argument is specified, then it
should be one of the flags described below. In this case the current
value of that setting is returned. Otherwise, the options should be a
set of flags and values that define the configuration:

- **-accept **mimetypes
  The Accept header of the request. The default is */* , which
  means that all types of documents are accepted. Otherwise
  you can supply a comma-separated list of mime type patterns
  that you are willing to receive. For example, “image/gif,
  image/jpeg, text/*”.

- **-proxyhost hostname**
The name of the proxy host, if any. If this value is the empty
  string, the URL host is contacted directly.

- **-proxypport number**
The proxy port number.

- **-proxyfilter command**
The command is a callback that is made during `::http::geturl`
to determine if a proxy is required for a given host. One
argument, a host name, is added to command when it is
invoked. If a proxy is required, the callback should return a
two-element list containing the proxy server and proxy port.
Otherwise the filter should return an empty list. The default
filter returns the values of the -proxyhost and -proxypport
settings if they are non-empty.

-urlencoding encoding
The encoding used for creating the x-url-encoded URLS with
::http::formatQuery. The default is utf-8, as specified by RFC
2718. Prior to http 2.5 this was unspecified, and that behavior
can be returned by specifying the empty string ({}), although
iso8859-1 is recommended to restore similar behavior but
without the ::http::formatQuery throwing an error processing
non-latin-1 characters.

-useragent string
The value of the User-Agent header in the HTTP request. The
default is “Tcl http client package 2.7”.

::http::geturl url ?options?
The ::http::geturl command is the main procedure in the package.
The -query option causes a POST operation and the -validate
option causes a HEAD operation; otherwise, a GET operation is
performed. The ::http::geturl command returns a token value that
can be used to get information about the transaction. See the
STATE ARRAY and ERRORS section for details. The ::http::geturl
command blocks until the operation completes, unless the -
command option specifies a callback that is invoked when the
HTTP transaction completes. ::http::geturl takes several options:

-binary boolean
Specifies whether to force interpreting the URL data as binary.
Normally this is auto-detected (anything not beginning with a
text content type or whose content encoding is gzip or
compress is considered considered binary data).

-blocksize size
The block size used when reading the URL. At most size bytes are read at once. After each block, a call to the -progress callback is made (if that option is specified).

-channel name
Copy the URL contents to channel name instead of saving it in state(body).

-command callback
Invoke callback after the HTTP transaction completes. This option causes ::http::geturl to return immediately. The callback gets an additional argument that is the token returned from ::http::geturl. This token is the name of an array that is described in the STATE ARRAY section. Here is a template for the callback:

```tcl
proc httpCallback {token} {
    upvar #0 $token state
    # Access state as a Tcl array
}
```

-handler callback
Invoke callback whenever HTTP data is available; if present, nothing else will be done with the HTTP data. This procedure gets two additional arguments: the socket for the HTTP data and the token returned from ::http::geturl. The token is the name of a global array that is described in the STATE ARRAY section. The procedure is expected to return the number of bytes read from the socket. Here is a template for the callback:

```tcl
proc httpHandlerCallback {socket token} {
    upvar #0 $token state
    # Access socket, and state as a Tcl array
    # For example...
    ...
```
set data [read $socket 1000]
set nbytes [string length $data]
...
return $nbytes

=headers keyvaluelist
This option is used to add extra headers to the HTTP request. The keyvaluelist argument must be a list with an even number of elements that alternate between keys and values. The keys become header field names. Newlines are stripped from the values so the header cannot be corrupted. For example, if keyvaluelist is Pragma no-cache then the following header is included in the HTTP request:

Pragma: no-cache

-keepalive boolean
If true, attempt to keep the connection open for servicing multiple requests. Default is 0.

-method type
Force the HTTP request method to type. ::http::geturl will auto-select GET, POST or HEAD based on other options, but this option enables choices like PUT and DELETE for webdav support.

-myaddr address
Pass an specific local address to the underlying socket call in case multiple interfaces are available.

-progress callback
The callback is made after each transfer of data from the URL. The callback gets three additional arguments: the token from ::http::geturl, the expected total size of the contents from the
Content-Length meta-data, and the current number of bytes transferred so far. The expected total size may be unknown, in which case zero is passed to the callback. Here is a template for the progress callback:

```tcl
proc httpProgress {token total current} {
    upvar #0 $token state
}
```

-protocol version
Select the HTTP protocol version to use. This should be 1.0 or 1.1 (the default). Should only be necessary for servers that do not understand or otherwise complain about HTTP/1.1.

-query query
This flag causes ::http::geturl to do a POST request that passes the query to the server. The query must be an x-url-encoding formatted query. The ::http::formatQuery procedure can be used to do the formatting.

-queryblocksize size
The block size used when posting query data to the URL. At most size bytes are written at once. After each block, a call to the -queryprogress callback is made (if that option is specified).

-querychannel channelID
This flag causes ::http::geturl to do a POST request that passes the data contained in channelID to the server. The data contained in channelID must be an x-url-encoding formatted query unless the -type option below is used. If a Content-Length header is not specified via the -headers options, ::http::geturl attempts to determine the size of the post data in order to create that header. If it is unable to determine the size, it returns an error.
-queryprogress *callback*

The *callback* is made after each transfer of data to the URL (i.e. POST) and acts exactly like the -progress option (the callback format is the same).

-strict *boolean*

Whether to enforce RFC 3986 URL validation on the request. Default is 1.

-timeout *milliseconds*

If *milliseconds* is non-zero, then `::http::geturl` sets up a timeout to occur after the specified number of milliseconds. A timeout results in a call to `::http::reset` and to the -command callback, if specified. The return value of `::http::status` is timeout after a timeout has occurred.

-type *mime-type*

Use *mime-type* as the Content-Type value, instead of the default value (*application/x-www-form-urlencoded*) during a POST operation.

-validate *boolean*

If *boolean* is non-zero, then `::http::geturl` does an HTTP HEAD request. This request returns meta information about the URL, but the contents are not returned. The meta information is available in the state(meta) variable after the transaction. See the STATE ARRAY section for details.

`::http::formatQuery` *key value <key value> ...?*

This procedure does x-url-encoding of query data. It takes an even number of arguments that are the keys and values of the query. It encodes the keys and values, and generates one string that has the proper & and = separators. The result is suitable for the -query value passed to `::http::geturl`.

`::http::reset` *token <why>*

This command resets the HTTP transaction identified by *token*, if any. This sets the state(status) value to *why*, which defaults to
reset, and then calls the registered -command callback.

::http::wait token
This is a convenience procedure that blocks and waits for the transaction to complete. This only works in trusted code because it uses vwait. Also, it is not useful for the case where ::http::geturl is called without the -command option because in this case the ::http::geturl call does not return until the HTTP transaction is complete, and thus there is nothing to wait for.

::http::data token
This is a convenience procedure that returns the body element (i.e., the URL data) of the state array.

::http::error token
This is a convenience procedure that returns the error element of the state array.

::http::status token
This is a convenience procedure that returns the status element of the state array.

::http::code token
This is a convenience procedure that returns the http element of the state array.

::http::ncode token
This is a convenience procedure that returns just the numeric return code (200, 404, etc.) from the http element of the state array.

::http::size token
This is a convenience procedure that returns the currentsize element of the state array, which represents the number of bytes received from the URL in the ::http::geturl call.

::http::meta token
This is a convenience procedure that returns the meta element of
the state array which contains the HTTP response headers. See below for an explanation of this element.

::http::cleanup token
This procedure cleans up the state associated with the connection identified by token. After this call, the procedures like ::http::data cannot be used to get information about the operation. It is strongly recommended that you call this function after you are done with a given HTTP request. Not doing so will result in memory not being freed, and if your app calls ::http::geturl enough times, the memory leak could cause a performance hit...or worse.

::http::register proto port command
This procedure allows one to provide custom HTTP transport types such as HTTPS, by registering a prefix, the default port, and the command to execute to create the Tcl channel. E.g.:

```
package require http
package require tls
::http::register https 443 ::tls::socket
set token [::http::geturl https://my.secure.site/]
```

::http::unregister proto
This procedure unregisters a protocol handler that was previously registered via ::http::register.

**ERRORS**

The ::http::geturl procedure will raise errors in the following cases: invalid command line options, an invalid URL, a URL on a non-existent host, or a URL at a bad port on an existing host. These errors mean that it cannot even start the network transaction. It will also raise an error if it gets an I/O error while writing out the HTTP request header. For synchronous ::http::geturl calls (where -command is not specified), it
will raise an error if it gets an I/O error while reading the HTTP reply headers or data. Because ::http::geturl does not return a token in these cases, it does all the required cleanup and there is no issue of your app having to call ::http::cleanup.

For asynchronous ::http::geturl calls, all of the above error situations apply, except that if there is any error while reading the HTTP reply headers or data, no exception is thrown. This is because after writing the HTTP headers, ::http::geturl returns, and the rest of the HTTP transaction occurs in the background. The command callback can check if any error occurred during the read by calling ::http::status to check the status and if its error, calling ::http::error to get the error message.

Alternatively, if the main program flow reaches a point where it needs to know the result of the asynchronous HTTP request, it can call ::http::wait and then check status and error, just as the callback does.

In any case, you must still call ::http::cleanup to delete the state array when you are done.

There are other possible results of the HTTP transaction determined by examining the status from ::http::status. These are described below.

ok
If the HTTP transaction completes entirely, then status will be ok. However, you should still check the ::http::code value to get the HTTP status. The ::http::ncode procedure provides just the numeric error (e.g., 200, 404 or 500) while the ::http::code procedure returns a value like “HTTP 404 File not found”.

eof
If the server closes the socket without replying, then no error is raised, but the status of the transaction will be eof.

error
The error message will also be stored in the error status array element, accessible via ::http::error.
Another error possibility is that ::http::geturl is unable to write all the post query data to the server before the server responds and closes the socket. The error message is saved in the posterror status array element and then ::http::geturl attempts to complete the transaction. If it can read the server's response it will end up with an ok status, otherwise it will have an eof status.

STATE ARRAY

The ::http::geturl procedure returns a token that can be used to get to the state of the HTTP transaction in the form of a Tcl array. Use this construct to create an easy-to-use array variable:

```
upvar #0 $token state
```

Once the data associated with the URL is no longer needed, the state array should be unset to free up storage. The ::http::cleanup procedure is provided for that purpose. The following elements of the array are supported:

**body**

The contents of the URL. This will be empty if the -channel option has been specified. This value is returned by the ::http::data command.

**charset**

The value of the charset attribute from the Content-Type meta-data value. If none was specified, this defaults to the RFC standard iso8859-1, or the value of $::http::defaultCharset. Incoming text data will be automatically converted from this charset to utf-8.

**coding**

A copy of the Content-Encoding meta-data value.

**currentsize**

The current number of bytes fetched from the URL. This value is returned by the ::http::size command.
error
If defined, this is the error string seen when the HTTP transaction was aborted.

http
The HTTP status reply from the server. This value is returned by the ::http::code command. The format of this value is:

HTTP/1.1 code string

The code is a three-digit number defined in the HTTP standard. A code of 200 is OK. Codes beginning with 4 or 5 indicate errors. Codes beginning with 3 are redirection errors. In this case the Location meta-data specifies a new URL that contains the requested information.

meta
The HTTP protocol returns meta-data that describes the URL contents. The meta element of the state array is a list of the keys and values of the meta-data. This is in a format useful for initializing an array that just contains the meta-data:

array set meta $state(meta)

Some of the meta-data keys are listed below, but the HTTP standard defines more, and servers are free to add their own.

Content-Type
The type of the URL contents. Examples include text/html, image/gif, application/postscript and application/x-tcl.

Content-Length
The advertised size of the contents. The actual size obtained by ::http::geturl is available as state(size).
**Location**

An alternate URL that contains the requested data.

**posterror**

The error, if any, that occurred while writing the post query data to the server.

**status**

Either **ok**, for successful completion, **reset** for user-reset, **timeout** if a timeout occurred before the transaction could complete, or **error** for an error condition. During the transaction this value is the empty string.

**totalsize**

A copy of the **Content-Length** meta-data value.

**type**

A copy of the **Content-Type** meta-data value.

**url**

The requested URL.

---

**EXAMPLE**

```bash
# Copy a URL to a file and print meta-data
proc httpcopy { url file {chunk 4096} } {
    set out [open $file w]    
    set token [::http::geturl $url -channel $out \    
               -progress httpCopyProgress -blocksize $chunk]
    close $out

    # This ends the line started by httpCopyProgress
    puts stderr ""

    upvar #0 $token state
    set max 0
    foreach {name value} $state(meta) {
```
if {[string length $name] > $max} {
    set max [string length $name]
}
if {[regexp -nocase ^location$ $name]} {
    # Handle URL redirects
    puts stderr "Location:$value"
    return [httpcopy [string trim $value] $file]
}
incr max
foreach {name value} $state(meta) {
    puts [format "%-*s %s" $max $name: $value]
}
return $token
}
proc httpCopyProgress {args} {
    puts -nonewline stderr .
    flush stderr
}
NAME
open - Open a file-based or command pipeline channel

SYNOPSIS
DESCRIPTION

r
r+

w
w+

a
a+

RDONLY
WRONLY
RDWR
APPEND
BINARY
CREAT
EXCL
NOCTTY
NONBLOCK
TRUNC

COMMAND PIPELINES
SERIAL COMMUNICATIONS

-mode baud,parity,data,stop
-handshake type
-queue
-timeout msec
-ttycontrol {signal boolean signal boolean ...}
-ttystatus
-xchar {xonChar xoffChar}
-pollinterval msec
-sysbuffer inSize
open - Open a file-based or command pipeline channel

**SYNOPSIS**

open fileName
open fileName access
open fileName access permissions
DESCRIPTION

This command opens a file, serial port, or command pipeline and returns a channel identifier that may be used in future invocations of commands like **read**, **puts**, and **close**. If the first character of **fileName** is not | then the command opens a file: **fileName** gives the name of the file to open, and it must conform to the conventions described in the **filename** manual entry.

The access argument, if present, indicates the way in which the file (or command pipeline) is to be accessed. In the first form **access** may have any of the following values:

- **r**
  
  Open the file for reading only; the file must already exist. This is the default value if **access** is not specified.

- **r+**
  
  Open the file for both reading and writing; the file must already exist.

- **w**
  
  Open the file for writing only. Truncate it if it exists. If it does not exist, create a new file.

- **w+**
  
  Open the file for reading and writing. Truncate it if it exists. If it does not exist, create a new file.

- **a**
  
  Open the file for writing only. If the file does not exist, create a new empty file. Set the file pointer to the end of the file prior to each write.

- **a+**
  
  Open the file for reading and writing. If the file does not exist, create a new empty file. Set the initial access position to the end of the file.
All of the legal access values above may have the character \texttt{b} added as the second or third character in the value to indicate that the opened channel should be configured with the \texttt{-translation binary} option, making the channel suitable for reading or writing of binary data.

In the second form, access consists of a list of any of the following flags, all of which have the standard POSIX meanings. One of the flags must be either \texttt{RDONLY}, \texttt{WRONLY} or \texttt{RDWR}.

**RDONLY**
Open the file for reading only.

**WRONLY**
Open the file for writing only.

**RDWR**
Open the file for both reading and writing.

**APPEND**
Set the file pointer to the end of the file prior to each write.

**BINARY**
Configure the opened channel with the \texttt{-translation binary} option.

**CREAT**
Create the file if it does not already exist (without this flag it is an error for the file not to exist).

**EXCL**
If \texttt{CREAT} is also specified, an error is returned if the file already exists.

**NOCTTY**
If the file is a terminal device, this flag prevents the file from becoming the controlling terminal of the process.

**NONBLOCK**
Prevents the process from blocking while opening the file, and possibly in subsequent I/O operations. The exact behavior of this
flag is system- and device-dependent; its use is discouraged (it is better to use the `fconfigure` command to put a file in nonblocking mode). For details refer to your system documentation on the `open` system call's `O_NONBLOCK` flag.

**TRUNC**
If the file exists it is truncated to zero length.

If a new file is created as part of opening it, `permissions` (an integer) is used to set the permissions for the new file in conjunction with the process's file mode creation mask. Permissions defaults to 0666.

**COMMAND PIPELINES**

If the first character of `fileName` is “|” then the remaining characters of `fileName` are treated as a list of arguments that describe a command pipeline to invoke, in the same style as the arguments for `exec`. In this case, the channel identifier returned by `open` may be used to write to the command's input pipe or read from its output pipe, depending on the value of `access`. If write-only access is used (e.g. `access` is `w`), then standard output for the pipeline is directed to the current standard output unless overridden by the command. If read-only access is used (e.g. `access` is `r`), standard input for the pipeline is taken from the current standard input unless overridden by the command. The id of the spawned process is accessible through the `pid` command, using the channel id returned by `open` as argument.

If the command (or one of the commands) executed in the command pipeline returns an error (according to the definition in `exec`), a Tcl error is generated when `close` is called on the channel unless the pipeline is in non-blocking mode then no exit status is returned (a silent `close` with -blocking 0).

It is often useful to use the `fileevent` command with pipelines so other processing may happen at the same time as running the command in the background.

**SERIAL COMMUNICATIONS**
If `fileName` refers to a serial port, then the specified serial port is opened and initialized in a platform-dependent manner. Acceptable values for the `fileName` to use to open a serial port are described in the PORTABILITY ISSUES section.

The `fconfigure` command can be used to query and set additional configuration options specific to serial ports (where supported):

**-mode baud,parity,data,stop**
This option is a set of 4 comma-separated values: the baud rate, parity, number of data bits, and number of stop bits for this serial port. The `baud` rate is a simple integer that specifies the connection speed. `Parity` is one of the following letters: `n`, `o`, `e`, `m`, `s`; respectively signifying the parity options of “none”, “odd”, “even”, “mark”, or “space”. `Data` is the number of data bits and should be an integer from 5 to 8, while `stop` is the number of stop bits and should be the integer 1 or 2.

**-handshake type**
(Windows and Unix). This option is used to setup automatic handshake control. Note that not all handshake types maybe supported by your operating system. The `type` parameter is case-independent.

If `type` is `none` then any handshake is switched off. `rtscts` activates hardware handshake. Hardware handshake signals are described below. For software handshake `xonxoff` the handshake characters can be redefined with `-xchar`. An additional hardware handshake `dtrdsr` is available only under Windows. There is no default handshake configuration, the initial value depends on your operating system settings. The `-handshake` option cannot be queried.

**-queue**
(Windows and Unix). The `-queue` option can only be queried. It returns a list of two integers representing the current number of bytes in the input and output queue respectively.
-timeout msec
(Windows and Unix). This option is used to set the timeout for blocking read operations. It specifies the maximum interval between the reception of two bytes in milliseconds. For Unix systems the granularity is 100 milliseconds. The -timeout option does not affect write operations or nonblocking reads. This option cannot be queried.

-ttycontrol {signal boolean signal boolean ...}
(Windows and Unix). This option is used to setup the handshake output lines (see below) permanently or to send a BREAK over the serial line. The signal names are case-independent. {RTS 1 DTR 0} sets the RTS output to high and the DTR output to low. The BREAK condition (see below) is enabled and disabled with {BREAK 1} and {BREAK 0} respectively. It is not a good idea to change the RTS (or DTR) signal with active hardware handshake rtscts (or dtrdsr). The result is unpredictable. The -ttycontrol option cannot be queried.

-ttystatus
(Windows and Unix). The -ttystatus option can only be queried. It returns the current modem status and handshake input signals (see below). The result is a list of signal,value pairs with a fixed order, e.g. {CTS 1 DSR 0 RING 1 DCD 0}. The signal names are returned upper case.

-xchar {xonChar xoffChar}
(Windows and Unix). This option is used to query or change the software handshake characters. Normally the operating system default should be DC1 (0x11) and DC3 (0x13) representing the ASCII standard XON and XOFF characters.

-pollinterval msec
(Windows only). This option is used to set the maximum time between polling for fileevents. This affects the time interval between checking for events throughout the Tcl interpreter (the smallest value always wins). Use this option only if you want to poll the serial port more or less often than 10 msec (the default).
-sysbuffer \textit{inSize}

-\textbf{sysbuffer} \{\textit{inSize outSize}\}

(Windows only). This option is used to change the size of Windows system buffers for a serial channel. Especially at higher communication rates the default input buffer size of 4096 bytes can overrun for latent systems. The first form specifies the input buffer size, in the second form both input and output buffers are defined.

\textbf{-lasterror}

(Windows only). This option is query only. In case of a serial communication error, \texttt{read} or \texttt{puts} returns a general Tcl file I/O error. \texttt{fconfigure -lasterror} can be called to get a list of error details. See below for an explanation of the various error codes.

\begin{table}[h]
\centering
\caption{Serial Port Signals}
\begin{tabular}{l l}
\hline
\textbf{TXD(output)} & Transmitted Data: Outgoing serial data. \\
\hline
\textbf{RXD(input)} & Received Data:Incoming serial data. \\
\hline
\textbf{RTS(output)} & Request To Send: This hardware handshake line informs the modem that your workstation is ready to receive data. Your workstation may automatically reset this signal to indicate that the input buffer is full. \\
\hline
\end{tabular}
\end{table}

RS-232 is the most commonly used standard electrical interface for serial communications. A negative voltage (-3V..-12V) define a mark (on=1) bit and a positive voltage (+3..+12V) define a space (off=0) bit (RS-232C). The following signals are specified for incoming and outgoing data, status lines and handshaking. Here we are using the terms \textit{workstation} for your computer and \textit{modem} for the external device, because some signal names (DCD, RI) come from modems. Of course your external device may use these signal lines for other purposes.
CTS(input)

Clear To Send: The complement to RTS. Indicates that the modem is ready to receive data.

DTR(output)

Data Terminal Ready: This signal tells the modem that the workstation is ready to establish a link. DTR is often enabled automatically whenever a serial port is opened.

DSR(input)

Data Set Ready: The complement to DTR. Tells the workstation that the modem is ready to establish a link.

DCD(input)

Data Carrier Detect: This line becomes active when a modem detects a “Carrier” signal.

RI(input)

Ring Indicator: Goes active when the modem detects an incoming call.

BREAK

A BREAK condition is not a hardware signal line, but a logical zero on the TXD or RXD lines for a long period of time, usually 250 to 500 milliseconds. Normally a receive or transmit data signal stays at the mark (on=1) voltage until the next character is transferred. A BREAK is sometimes used to reset the communications line or change the operating mode of communications hardware.

ERROR CODES (Windows only)

A lot of different errors may occur during serial read operations or during event polling in background. The external device may have been switched off, the data lines may be noisy, system buffers may overrun or your mode settings may be wrong. That is why a reliable software should always catch serial read operations. In cases of an error Tcl returns a general file I/O error. Then `fconfigure -lasterror` may help to locate the problem. The following error codes may be returned.
RXOVER
Windows input buffer overrun. The data comes faster than your scripts reads it or your system is overloaded. Use `fconfigure -sysbuffer` to avoid a temporary bottleneck and/or make your script faster.

TXFULL
Windows output buffer overrun. Complement to RXOVER. This error should practically not happen, because Tcl cares about the output buffer status.

OVERRUN
UART buffer overrun (hardware) with data lost. The data comes faster than the system driver receives it. Check your advanced serial port settings to enable the FIFO (16550) buffer and/or setup a lower(1) interrupt threshold value.

RXPARITY
A parity error has been detected by your UART. Wrong parity settings with `fconfigure -mode` or a noisy data line (RXD) may cause this error.

FRAME
A stop-bit error has been detected by your UART. Wrong mode settings with `fconfigure -mode` or a noisy data line (RXD) may cause this error.

BREAK
A BREAK condition has been detected by your UART (see above).

PORTABILITY ISSUES

Windows (all versions)
Valid values for `fileName` to open a serial port are of the form `comX:`, where X is a number, generally from 1 to 4. This notation only works for serial ports from 1 to 9, if the system happens to have more than four. An attempt to open a serial port that does not exist or has a number greater than 9 will fail. An alternate form of
opening serial ports is to use the filename `\\.com\X`, where X is any number that corresponds to a serial port; please note that this method is considerably slower on Windows 95 and Windows 98.

**Windows NT**

When running Tcl interactively, there may be some strange interactions between the real console, if one is present, and a command pipeline that uses standard input or output. If a command pipeline is opened for reading, some of the lines entered at the console will be sent to the command pipeline and some will be sent to the Tcl evaluator. If a command pipeline is opened for writing, keystrokes entered into the console are not visible until the pipe is closed. This behavior occurs whether the command pipeline is executing 16-bit or 32-bit applications. These problems only occur because both Tcl and the child application are competing for the console at the same time. If the command pipeline is started from a script, so that Tcl is not accessing the console, or if the command pipeline does not use standard input or output, but is redirected from or to a file, then the above problems do not occur.

**Windows 95**

A command pipeline that executes a 16-bit DOS application cannot be opened for both reading and writing, since 16-bit DOS applications that receive standard input from a pipe and send standard output to a pipe run synchronously. Command pipelines that do not execute 16-bit DOS applications run asynchronously and can be opened for both reading and writing.

When running Tcl interactively, there may be some strange interactions between the real console, if one is present, and a command pipeline that uses standard input or output. If a command pipeline is opened for reading from a 32-bit application, some of the keystrokes entered at the console will be sent to the command pipeline and some will be sent to the Tcl evaluator. If a command pipeline is opened for writing to a 32-bit application, no output is visible on the console until the pipe is closed. These problems only occur because both Tcl and the child application are competing for the console at the same time. If the command pipeline is started
from a script, so that Tcl is not accessing the console, or if the command pipeline does not use standard input or output, but is redirected from or to a file, then the above problems do not occur.

Whether or not Tcl is running interactively, if a command pipeline is opened for reading from a 16-bit DOS application, the call to `open` will not return until end-of-file has been received from the command pipeline's standard output. If a command pipeline is opened for writing to a 16-bit DOS application, no data will be sent to the command pipeline's standard output until the pipe is actually closed. This problem occurs because 16-bit DOS applications are run synchronously, as described above.

**Unix**

Valid values for `fileName` to open a serial port are generally of the form `/dev/ttyX`, where `X` is `a` or `b`, but the name of any pseudo-file that maps to a serial port may be used. Advanced configuration options are only supported for serial ports when Tcl is built to use the POSIX serial interface.

When running Tcl interactively, there may be some strange interactions between the console, if one is present, and a command pipeline that uses standard input. If a command pipeline is opened for reading, some of the lines entered at the console will be sent to the command pipeline and some will be sent to the Tcl evaluator. This problem only occurs because both Tcl and the child application are competing for the console at the same time. If the command pipeline is started from a script, so that Tcl is not accessing the console, or if the command pipeline does not use standard input, but is redirected from a file, then the above problem does not occur.

See the **PORTABILITY ISSUES** section of the `exec` command for additional information not specific to command pipelines about executing applications on the various platforms.

**EXAMPLE**
Open a command pipeline and catch any errors:

```
set fl [open "| ls this_file_does_not_exist"]
set data [read $fl]
if {{[catch {close $fl} err]} {puts "ls command failed: $err"}}
```

**SEE ALSO**

file, close, filename, fconfigure, gets, read, puts, exec, pid, fopen

**KEYWORDS**

access mode, append, create, file, non-blocking, open, permissions, pipeline, process, serial

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

split - Split a string into a proper Tcl list

SYNOPSIS

split string ?splitChars?

DESCRIPTION

Returns a list created by splitting string at each character that is in the splitChars argument. Each element of the result list will consist of the characters from string that lie between instances of the characters in splitChars. Empty list elements will be generated if string contains adjacent characters in splitChars, or if the first or last character of string is in splitChars. If splitChars is an empty string then each character of string becomes a separate element of the result list. SplitChars defaults to the standard white-space characters.

EXAMPLES

Divide up a USENET group name into its hierarchical components:

```
    split "comp.lang.tcl.announce" .
    → comp lang tcl announce
```

See how the split command splits on every character in splitChars, which can result in information loss if you are not careful:
```
split "alpha beta gamma" "temp"
→ al {ha b} {} {a ga} {} a
```

Extract the list words from a string that is not a well-formed list:

```
split "Example with {unbalanced brace character"
→ Example with \{unbalanced brace character
```

Split a string into its constituent characters

```
split "Hello world" {}
→ H e l l o { } w o r l d
```

### PARSING RECORD-ORIENTED FILES

Parse a Unix `/etc/passwd` file, which consists of one entry per line, with each line consisting of a colon-separated list of fields:

```
## Read the file
set fid [open /etc/passwd]
set content [read $fid]
close $fid

## Split into records on newlines
set records [split $content "\n"]

## Iterate over the records
foreach rec $records {

## Split into fields on colons
set fields [split $rec ":"]
```
## Assign fields to variables and print some out.

lassign $fields \
        userName password uid grp longName homeDir
puts "$longName uses [file tail $shell] for a log"
while - Execute script repeatedly as long as a condition is met

SYNOPSIS

while test body

DESCRIPTION

The while command evaluates test as an expression (in the same way that expr evaluates its argument). The value of the expression must a proper boolean value; if it is a true value then body is executed by passing it to the Tcl interpreter. Once body has been executed then test is evaluated again, and the process repeats until eventually test evaluates to a false boolean value. Continue commands may be executed inside body to terminate the current iteration of the loop, and break commands may be executed inside body to cause immediate termination of the while command. The while command always returns an empty string.

Note: test should almost always be enclosed in braces. If not, variable substitutions will be made before the while command starts executing, which means that variable changes made by the loop body will not be considered in the expression. This is likely to result in an infinite loop. If test is enclosed in braces, variable substitutions are delayed until the expression is evaluated (before each loop iteration), so changes in the variables will be visible. For an example, try the following script with and without the braces around $x<10$:

```tcl
set x 0
```
while {$x<10} {
  puts "x is $x"
  incr x
}

EXAMPLE

Read lines from a channel until we get to the end of the stream, and print them out with a line-number prepended:

set lineCount 0
while {{[gets $chan line] >= 0} {
  puts "[incr lineCount]: $line"
}

SEE ALSO

break, continue, for, foreach

KEYWORDS

boolean value, loop, test, while

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME

continue - Skip to the next iteration of a loop

SYNOPSIS

continue

DESCRIPTION

This command is typically invoked inside the body of a looping command such as for or foreach or while. It returns a TCL_CONTINUE code, which causes a continue exception to occur. The exception causes the current script to be aborted out to the innermost containing loop command, which then continues with the next iteration of the loop. Catch exceptions are also handled in a few other situations, such as the catch command and the outermost scripts of procedure bodies.

EXAMPLE

Print a line for each of the integers from 0 to 10 except 5:

```tcl
for {set x 0} {$x<10} {incr x} {
    if {$x == 5} {
        continue
    }
    puts "x is $x"
}
```
SEE ALSO

break, for, foreach, return, while

KEYWORDS

continue, iteration, loop

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
**NAME**

if - Execute scripts conditionally

**SYNOPSIS**

```
if expr1 ?then? body1 elseif expr2 ?then? body2 elseif ... ?else? ?bodyN?
```

**DESCRIPTION**

The *if* command evaluates *expr1* as an expression (in the same way that *expr* evaluates its argument). The value of the expression must be a boolean (a numeric value, where 0 is false and anything is true, or a string value such as *true* or *yes* for true and *false* or *no* for false); if it is true then *body1* is executed by passing it to the Tcl interpreter. Otherwise *expr2* is evaluated as an expression and if it is true then *body2* is executed, and so on. If none of the expressions evaluates to true then *bodyN* is executed. The **then** and **else** arguments are optional “noise words” to make the command easier to read. There may be any number of elseif clauses, including zero. *BodyN* may also be omitted as long as **else** is omitted too. The return value from the command is the result of the body script that was executed, or an empty string if none of the expressions was non-zero and there was no *bodyN*.

**EXAMPLES**

A simple conditional:

```tcl
if {$vbl == 1} { puts "vbl is one" }
```
With an `else`-clause:

```bash
if {$vbl == 1} {
  puts "vbl is one"
} else {
  puts "vbl is not one"
}
```

With an `elseif`-clause too:

```bash
if {$vbl == 1} {
  puts "vbl is one"
} elseif {$vbl == 2} {
  puts "vbl is two"
} else {
  puts "vbl is not one or two"
}
```

Remember, expressions can be multi-line, but in that case it can be a good idea to use the optional `then` keyword for clarity:

```bash
if {
  $vbl == 1 || $vbl == 2 || $vbl == 3
} then {
  puts "vbl is one, two or three"
}
```

**SEE ALSO**

`expr`, `for`, `foreach`

**KEYWORDS**
boolean, conditional, else, false, if, true
NAME

package - Facilities for package loading and version control

SYNOPSIS

DESCRIPTION

package forget ?package package ...?
package ifneeded package version ?script?
package names
package present
package provide package ?version?
package require package ?requirement...?
package require -exact package version
package unknown ?command?
package vcompare version1 version2
package versions package
package vsatisfies version requirement...

min
min-
min-max
[a]
[b]
package prefer ?!latest|stable?

VERSION NUMBERS
PACKAGE INDICES
EXAMPLES
SEE ALSO
KEYWORDS

NAME

package - Facilities for package loading and version control
SYNOPSIS

package forget ?package package ...?
package ifneeded package version ?script?
package names
package present package ?requirement...?
package present -exact package version
package provide package ?version?
package require package ?requirement...?
package require -exact package version
package unknown ?command?
package vcompare version1 version2
package versions package
package vsatisfies version requirement...
package prefer ?latest|stable?

DESCRIPTION

This command keeps a simple database of the packages available for use by the current interpreter and how to load them into the interpreter. It supports multiple versions of each package and arranges for the correct version of a package to be loaded based on what is needed by the application. This command also detects and reports version clashes. Typically, only the package require and package provide commands are invoked in normal Tcl scripts; the other commands are used primarily by system scripts that maintain the package database.

The behavior of the package command is determined by its first argument. The following forms are permitted:

package forget ?package package ...?
   Removes all information about each specified package from this interpreter, including information provided by both package ifneeded and package provide.

package ifneeded package version ?script?
   This command typically appears only in system configuration scripts to set up the package database. It indicates that a particular
version of a particular package is available if needed, and that the package can be added to the interpreter by executing `script`. The script is saved in a database for use by subsequent `package require` commands; typically, `script` sets up auto-loading for the commands in the package (or calls `load` and/or `source` directly), then invokes `package provide` to indicate that the package is present. There may be information in the database for several different versions of a single package. If the database already contains information for `package` and `version`, the new `script` replaces the existing one. If the `script` argument is omitted, the current `script` for version `version` of package `package` is returned, or an empty string if no `package ifneeded` command has been invoked for this `package` and `version`.

**package names**
Returns a list of the names of all packages in the interpreter for which a version has been provided (via `package provide`) or for which a `package ifneeded` script is available. The order of elements in the list is arbitrary.

**package present**
This command is equivalent to `package require` except that it does not try and load the package if it is not already loaded.

**package provide package ?version?**
This command is invoked to indicate that version `version` of package `package` is now present in the interpreter. It is typically invoked once as part of an `ifneeded` script, and again by the package itself when it is finally loaded. An error occurs if a different version of `package` has been provided by a previous `package provide` command. If the `version` argument is omitted, then the command returns the version number that is currently provided, or an empty string if no `package provide` command has been invoked for `package` in this interpreter.

**package require package ?requirement...?**
This command is typically invoked by Tcl code that wishes to use a particular version of a particular package. The arguments indicate
which package is wanted, and the command ensures that a suitable version of the package is loaded into the interpreter. If the command succeeds, it returns the version number that is loaded; otherwise it generates an error.

A suitable version of the package is any version which satisfies at least one of the requirements, per the rules of package `vsatisfies`. If multiple versions are suitable the implementation with the highest version is chosen. This last part is additionally influenced by the selection mode set with `package prefer`.

In the “stable” selection mode the command will select the highest stable version satisfying the requirements, if any. If no stable version satisfies the requirements, the highest unstable version satisfying the requirements will be selected. In the “latest” selection mode the command will accept the highest version satisfying all the requirements, regardless of its stableness.

If a version of `package` has already been provided (by invoking the `package provide` command), then its version number must satisfy the `requirements` and the command returns immediately. Otherwise, the command searches the database of information provided by previous `package ifneeded` commands to see if an acceptable version of the package is available. If so, the script for the highest acceptable version number is evaluated in the global namespace; it must do whatever is necessary to load the package, including calling `package provide` for the package. If the `package ifneeded` database does not contain an acceptable version of the package and a `package unknown` command has been specified for the interpreter then that command is evaluated in the global namespace; when it completes, Tcl checks again to see if the package is now provided or if there is a `package ifneeded` script for it. If all of these steps fail to provide an acceptable version of the package, then the command returns an error.

`package require -exact package version`
This form of the command is used when only the given `version` of `package` is acceptable to the caller. This command is equivalent to
**package require** *package version-version.*

**package unknown** *?command?*
This command supplies a “last resort” command to invoke during **package require** if no suitable version of a package can be found in the **package ifneeded** database. If the *command* argument is supplied, it contains the first part of a command; when the command is invoked during a **package require** command, Tcl appends one or more additional arguments giving the desired package name and requirements. For example, if *command* is **foo bar** and later the command **package require test 2.4** is invoked, then Tcl will execute the command **foo bar test 2.4** to load the package. If no requirements are supplied to the **package require** command, then only the name will be added to invoked command. If the **package unknown** command is invoked without a *command* argument, then the current **package unknown** script is returned, or an empty string if there is none. If *command* is specified as an empty string, then the current **package unknown** script is removed, if there is one.

**package vcompare** *version1 version2*
Comparing the two version numbers given by *version1* and *version2*. Returns -1 if *version1* is an earlier version than *version2*, 0 if they are equal, and 1 if *version1* is later than *version2*.

**package versions** *package*
Returns a list of all the version numbers of *package* for which information has been provided by **package ifneeded** commands.

**package vsatisfies** *version requirement...*
Returns 1 if the *version* satisfies at least one of the given requirements, and 0 otherwise. Each *requirement* is allowed to have any of the forms:

- **min**
  This form is called “min-bounded”.

- **min**
This form is called “min-unbound”.

min-max

This form is called “bounded”.

where “min” and “max” are valid version numbers. The legacy syntax is a special case of the extended syntax, keeping backward compatibility. Regarding satisfaction the rules are:

[1]

The version has to pass at least one of the listed requirements to be satisfactory.

[2]

A version satisfies a “bounded” requirement when

[a]

For min equal to the max if, and only if the version is equal to the min.

[b]

Otherwise if, and only if the version is greater than or equal to the min, and less than the max, where both min and max have been padded internally with “a0”. Note that while the comparison to min is inclusive, the comparison to max is exclusive.

[3]

A “min-bounded” requirement is a “bounded” requirement in disguise, with the max part implicitly specified as the next higher major version number of the min part. A version satisfies it per the rules above.

[4]

A version satisfies a “min-unbound” requirement if, and only if it is greater than or equal to the min, where the min has been padded internally with “a0”. There is no constraint to a maximum.
package prefer ?{latest|stable}?

With no arguments, the commands returns either “latest” or “stable”, whichever describes the current mode of selection logic used by package require.

When passed the argument “latest”, it sets the selection logic mode to “latest”.

When passed the argument “stable”, if the mode is already “stable”, that value is kept. If the mode is already “latest”, then the attempt to set it back to “stable” is ineffective and the mode value remains “latest”.

When passed any other value as an argument, raise an invalid argument error.

When an interpreter is created, its initial selection mode value is set to “stable” unless the environment variable TCL_PKG_PREFER_LATEST is set. If that environment variable is defined (with any value) then the initial (and permanent) selection mode value is set to “latest”.

**VERSION NUMBERS**

Version numbers consist of one or more decimal numbers separated by dots, such as 2 or 1.162 or 3.1.13.1. The first number is called the major version number. Larger numbers correspond to later versions of a package, with leftmost numbers having greater significance. For example, version 2.1 is later than 1.3 and version 3.4.6 is later than 3.3.5. Missing fields are equivalent to zeroes: version 1.3 is the same as version 1.3.0 and 1.3.0.0, so it is earlier than 1.3.1 or 1.3.0.2. In addition, the letters “a” (alpha) and/or “b” (beta) may appear exactly once to replace a dot for separation. These letters semantically add a negative specifier into the version, where “a” is -2, and “b” is -1. Each may be specified only once, and “a” or “b” are mutually exclusive in a specifier. Thus 1.3a1 becomes (semantically) 1.3.-2.1, 1.3b1 is 1.3.-1.1. Negative numbers are not directly allowed in version specifiers. A version number not containing the letters “a” or “b” as specified above is
called a **stable** version, whereas presence of the letters causes the version to be called is **unstable**. A later version number is assumed to be upwards compatible with an earlier version number as long as both versions have the same major version number. For example, Tcl scripts written for version 2.3 of a package should work unchanged under versions 2.3.2, 2.4, and 2.5.1. Changes in the major version number signify incompatible changes: if code is written to use version 2.1 of a package, it is not guaranteed to work unmodified with either version 1.7.3 or version 3.1.

---

**PACKAGE INDICES**

The recommended way to use packages in Tcl is to invoke `package require` and `package provide` commands in scripts, and use the procedure `pkg_mkIndex` to create package index files. Once you have done this, packages will be loaded automatically in response to `package require` commands. See the documentation for `pkg_mkIndex` for details.

---

**EXAMPLES**

To state that a Tcl script requires the Tk and http packages, put this at the top of the script:

```tcl
package require Tk
package require http
```

To test to see if the Snack package is available and load if it is (often useful for optional enhancements to programs where the loss of the functionality is not critical) do this:

```tcl
if {[catch {package require Snack}]} {
    # Error thrown - package not found.
    # Set up a dummy interface to work around the abs
} else {
```
# We have the package, configure the app to use it.

## SEE ALSO

msgcat, packagens, pkgMkIndex

## KEYWORDS

package, version

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
NAME
  string - Manipulate strings

SYNOPSIS

DESCRIPTION

  string bytelength string
  string compare ?-nocase? ?-length int? string1 string2
  string equal ?-nocase? ?-length int? string1 string2
  string first needleString haystackString ?startIndex?
  string index string charIndex

  integer
  end
  end-N
  end+N
  M+N
  M-N

  string is class ?-strict? ?-failindex varname? string

  alnum
  alpha
  ascii
  boolean
  control
  digit
  double
  false
  graph
  integer
  list
  lower
  print
  punct
  space
true
upper
wideinteger
wordchar
xdigit

string last needleString haystackString ?lastIndex?
string length string
string map ?-nocase? mapping string
string match ?-nocase? pattern string

* 
? 
[chars]
\x

string range string first last
string repeat string count
string replace string first last ?newstring?
string reverse string
string tolower string ?first? ?last?
string totitle string ?first? ?last?
string toupper string ?first? ?last?
string trim string ?chars?
string trimleft string ?chars?
string trimright string ?chars?
string wordend string charIndex
string wordstart string charIndex

EXAMPLE
SEE ALSO
KEYWORDS

NAME

string - Manipulate strings

SYNOPSIS

string option arg ?arg ...?

DESCRIPTION
Performs one of several string operations, depending on option. The legal options (which may be abbreviated) are:

**string bytelen**

Returns a decimal string giving the number of bytes used to represent string in memory. Because UTF-8 uses one to three bytes to represent Unicode characters, the byte length will not be the same as the character length in general. The cases where a script cares about the byte length are rare. In almost all cases, you should use the string length operation (including determining the length of a Tcl ByteArray object). Refer to the *Tcl_NumUtfChars* manual entry for more details on the UTF-8 representation.

**string compare ?-nocase? ?-length int? string1 string2**

Perform a character-by-character comparison of strings string1 and string2. Returns -1, 0, or 1, depending on whether string1 is lexicographically less than, equal to, or greater than string2. If -length is specified, then only the first length characters are used in the comparison. If -length is negative, it is ignored. If -nocase is specified, then the strings are compared in a case-insensitive manner.

**string equal ?-nocase? ?-length int? string1 string2**

Perform a character-by-character comparison of strings string1 and string2. Returns 1 if string1 and string2 are identical, or 0 when not. If -length is specified, then only the first length characters are used in the comparison. If -length is negative, it is ignored. If -nocase is specified, then the strings are compared in a case-insensitive manner.

**string first needleString haystackString ?startIndex?**

Search haystackString for a sequence of characters that exactly match the characters in needleString. If found, return the index of the first character in the first such match within haystackString. If not found, return -1. If startIndex is specified (in any of the forms accepted by the index method), then the search is constrained to start with the character in haystackString specified by the index. For example,
string first a 0a23456789abcdef 5

will return 10, but

string first a 0123456789abcdef 11

will return -1.

**string index** string charIndex

Returns the charIndex'th character of the string argument. A charIndex of 0 corresponds to the first character of the string. charIndex may be specified as follows:

*integer*

For any index value that passes **string is integer -strict**, the char specified at this integral index (e.g. 2 would refer to the “c” in “abcd”).

*end*

The last char of the string (e.g. **end** would refer to the “d” in “abcd”).

*end-N*

The last char of the string minus the specified integer offset N (e.g. **end-1** would refer to the “c” in “abcd”).

*end+N*

The last char of the string plus the specified integer offset N (e.g. **end+-1** would refer to the “c” in “abcd”).

*M+N*

The char specified at the integral index that is the sum of integer values M and N (e.g. **1+1** would refer to the “c” in “abcd”).
The char specified at the integral index that is the difference of integer values $M$ and $N$ (e.g. `2-1` would refer to the “b” in “abcd”).

In the specifications above, the integer value $M$ contains no trailing whitespace and the integer value $N$ contains no leading whitespace.

If `charIndex` is less than 0 or greater than or equal to the length of the string then this command returns an empty string.

```
string is class ?-strict? ?-failindex varname? string
```

Returns 1 if `string` is a valid member of the specified character class, otherwise returns 0. If `-strict` is specified, then an empty string returns 0, otherwise an empty string will return 1 on any class. If `-failindex` is specified, then if the function returns 0, the index in the string where the class was no longer valid will be stored in the variable named `varname`. The `varname` will not be set if the function returns 1. The following character classes are recognized (the class name can be abbreviated):

- **alnum**
  Any Unicode alphabet or digit character.

- **alpha**
  Any Unicode alphabet character.

- **ascii**
  Any character with a value less than \u0080 (those that are in the 7-bit ascii range).

- **boolean**
  Any of the forms allowed to `Tcl_GetBoolean`.

- **control**
  Any Unicode control character.

- **digit**
Any Unicode digit character. Note that this includes characters outside of the [0-9] range.

**double**

Any of the valid forms for a double in Tcl, with optional surrounding whitespace. In case of under/overflow in the value, 0 is returned and the *varname* will contain -1.

**false**

Any of the forms allowed to [Tcl GetBoolean](https://www.tcl-lang.org/commands/tcl_getboolean) where the value is false.

**graph**

Any Unicode printing character, except space.

**integer**

Any of the valid string formats for a 32-bit integer value in Tcl, with optional surrounding whitespace. In case of under/overflow in the value, 0 is returned and the *varname* will contain -1.

**list**

Any proper list structure, with optional surrounding whitespace. In case of improper list structure, 0 is returned and the *varname* will contain the index of the “element” where the list parsing fails, or -1 if this cannot be determined.

**lower**

Any Unicode lower case alphabet character.

**print**

Any Unicode printing character, including space.

**punct**

Any Unicode punctuation character.

**space**

Any Unicode space character.
true
Any of the forms allowed to `Tcl_GetBoolean` where the value is true.

upper
Any upper case alphabet character in the Unicode character set.

wideinteger
Any of the valid forms for a wide integer in Tcl, with optional surrounding whitespace. In case of under/overflow in the value, 0 is returned and the `varname` will contain -1.

wordchar
Any Unicode word character. That is any alphanumeric character, and any Unicode connector punctuation characters (e.g. underscore).

xdigit
Any hexadecimal digit character ([0-9A-Fa-f]).

In the case of `boolean`, `true` and `false`, if the function will return 0, then the `varname` will always be set to 0, due to the varied nature of a valid boolean value.

**string last** `needleString haystackString` `?lastIndex`?
Search `haystackString` for a sequence of characters that exactly match the characters in `needleString`. If found, return the index of the first character in the last such match within `haystackString`. If there is no match, then return -1. If `lastIndex` is specified (in any of the forms accepted by the `index` method), then only the characters in `haystackString` at or before the specified `lastIndex` will be considered by the search. For example,

```
string last a 0a23456789abcdef 15
```

will return 10, but
will return 1.

**String Length**

*String length* *string*

Returns a decimal string giving the number of characters in *string*. Note that this is not necessarily the same as the number of bytes used to store the string. If the object is a ByteArray object (such as those returned from reading a binary encoded channel), then this will return the actual byte length of the object.

**String Map**

*String map ?-nocase? mapping string*

Replaces substrings in *string* based on the key-value pairs in *mapping*. *mapping* is a list of key value key value ... as in the form returned by *array get*. Each instance of a key in the string will be replaced with its corresponding value. If **-nocase** is specified, then matching is done without regard to case differences. Both *key* and *value* may be multiple characters. Replacement is done in an ordered manner, so the key appearing first in the list will be checked first, and so on. *string* is only iterated over once, so earlier key replacements will have no affect for later key matches. For example,

```
string map {abc 1 ab 2 a 3 1 0} 1abcaababcabababc
```

will return the string **01321221**.

Note that if an earlier *key* is a prefix of a later one, it will completely mask the later one. So if the previous example is reordered like this,

```
string map {1 0 ab 2 a 3 abc 1} 1abcaababcabababc
```

will produce the same result.
it will return the string 02c322c222c.

**string match ?-nocase? pattern string**
See if pattern matches string; return 1 if it does, 0 if it does not. If **nocase** is specified, then the pattern attempts to match against the string in a case insensitive manner. For the two strings to match, their contents must be identical except that the following special sequences may appear in pattern:

*  
  Matches any sequence of characters in string, including a null string.

?  
  Matches any single character in string.

[chars]  
  Matches any character in the set given by chars. If a sequence of the form x-y appears in chars, then any character between x and y, inclusive, will match. When used with **-nocase**, the end points of the range are converted to lower case first. Whereas {{[A-z]}} matches “_” when matching case-sensitively (since “_” falls between the “Z” and “a”), with **-nocase** this is considered like {{[A-Za-z]}} (and probably what was meant in the first place).

\x  
  Matches the single character x. This provides a way of avoiding the special interpretation of the characters *?[]\ in pattern.

**string range string first last**
Returns a range of consecutive characters from string, starting with the character whose index is first and ending with the character whose index is last. An index of 0 refers to the first character of the string. first and last may be specified as for the index method. If first is less than zero then it is treated as if it were zero, and if last is greater than or equal to the length of the string then it is treated as if it were end. If first is greater than last then an empty string is
string repeat string count
    Returns string repeated count number of times.

string replace string first last ?newstring?
    Removes a range of consecutive characters from string, starting
with the character whose index is first and ending with the
character whose index is last. An index of 0 refers to the first
character of the string. First and last may be specified as for the
index method. If newstring is specified, then it is placed in the
removed character range. If first is less than zero then it is treated
as if it were zero, and if last is greater than or equal to the length of
the string then it is treated as if it were end. If first is greater than
last or the length of the initial string, or last is less than 0, then the
initial string is returned untouched.

string reverse string
    Returns a string that is the same length as string but with its
characters in the reverse order.

string tolower string ?first? ?last?
    Returns a value equal to string except that all upper (or title) case
letters have been converted to lower case. If first is specified, it
refers to the first char index in the string to start modifying. If last is
specified, it refers to the char index in the string to stop at
(inclusive). first and last may be specified as for the index method.

string totitle string ?first? ?last?
    Returns a value equal to string except that the first character in
string is converted to its Unicode title case variant (or upper case if
there is no title case variant) and the rest of the string is converted
to lower case. If first is specified, it refers to the first char index in
the string to start modifying. If last is specified, it refers to the char
index in the string to stop at (inclusive). first and last may be
specified as for the index method.

string toupper string ?first? ?last?
Returns a value equal to \textit{string} except that all lower (or title) case letters have been converted to upper case. If \textit{first} is specified, it refers to the first char index in the string to start modifying. If \textit{last} is specified, it refers to the char index in the string to stop at (inclusive). \textit{first} and \textit{last} may be specified as for the \textbf{index} method.

\textbf{string trim} \textit{string} ?\textit{chars}?

Returns a value equal to \textit{string} except that any leading or trailing characters present in the string given by \textit{chars} are removed. If \textit{chars} is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

\textbf{string trimleft} \textit{string} ?\textit{chars}?

Returns a value equal to \textit{string} except that any leading characters present in the string given by \textit{chars} are removed. If \textit{chars} is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

\textbf{string trimright} \textit{string} ?\textit{chars}?

Returns a value equal to \textit{string} except that any trailing characters present in the string given by \textit{chars} are removed. If \textit{chars} is not specified then white space is removed (spaces, tabs, newlines, and carriage returns).

\textbf{string wordend} \textit{string} \textit{charIndex}

Returns the index of the character just after the last one in the word containing character \textit{charIndex} of \textit{string}. \textit{charIndex} may be specified as for the \textbf{index} method. A word is considered to be any contiguous range of alphanumeric (Unicode letters or decimal digits) or underscore (Unicode connector punctuation) characters, or any single character other than these.

\textbf{string wordstart} \textit{string} \textit{charIndex}

Returns the index of the first character in the word containing character \textit{charIndex} of \textit{string}. \textit{charIndex} may be specified as for the \textbf{index} method. A word is considered to be any contiguous range of alphanumeric (Unicode letters or decimal digits) or underscore (Unicode connector punctuation) characters, or any single
character other than these.

**EXAMPLE**

Test if the string in the variable `string` is a proper non-empty prefix of the string `foobar`.

```bash
set length [string length $string]
if {$length == 0} {
    set isPrefix 0
} else {
    set isPrefix [string equal -length $length $string]
}
```

**SEE ALSO**

`expr`, `list`

**KEYWORDS**

`case conversion`, `compare`, `index`, `match`, `pattern`, `string`, `word`, `equal`, `ctype`, `character`, `reverse`
NAME
dde - Execute a Dynamic Data Exchange command

SYNOPSIS

DESCRIPTION

DDE COMMANDS

dde servername ?-force? ?-handler proc? ?--? ?topic?
dde execute ?-async? service topic data
dde poke service topic item data
dde request ?-binary? service topic item
dde services service topic
dde eval ?-async? topic cmd ?arg arg ...

DDE AND TCL

EXAMPLE

SEE ALSO

KEYWORDS
This command allows an application to send Dynamic Data Exchange (DDE) command when running under Microsoft Windows. Dynamic Data Exchange is a mechanism where applications can exchange raw data. Each DDE transaction needs a service name and a topic. Both the service name and topic are application defined; Tcl uses the service name TclEval, while the topic name is the name of the interpreter given by dde servername. Other applications have their own service names and topics. For instance, Microsoft Excel has the service name Excel.

**DDE COMMANDS**

The following commands are a subset of the full Dynamic Data Exchange set of commands.

**dde servername**  

**dde servername** registers the interpreter as a DDE server with the service name TclEval and the topic name specified by topic. If no topic is given, **dde servername** returns the name of the current topic or the empty string if it is not registered as a service. If the given topic name is already in use, then a suffix of the form “ #2” or “ #3” is appended to the name to make it unique. The command's result will be the name actually used. The **-force** option is used to force registration of precisely the given topic name.

The **-handler** option specifies a Tcl procedure that will be called to process calls to the dde server. If the package has been loaded into a safe interpreter then a **-handler** procedure must be defined. The procedure is called with all the arguments provided by the remote call.

**dde execute**  
?-async? service topic data

**dde execute** takes the data and sends it to the server indicated by service with the topic indicated by topic. Typically, service is the name of an application, and topic is a file to work on. The data field is given to the remote application. Typically, the application treats the data field as a script, and the script is run in the application. The **-async** option requests asynchronous invocation. The command returns an error message if the script did not run, unless
the -async flag was used, in which case the command returns immediately with no error.

**dde poke** `service topic item data`  
**dde poke** passes the `data` to the server indicated by `service` using the `topic` and `item` specified. Typically, `service` is the name of an application. `topic` is application specific but can be a command to the server or the name of a file to work on. The `item` is also application specific and is often not used, but it must always be non-null. The `data` field is given to the remote application.

**dde request** `-binary? service topic item`  
**dde request** is typically used to get the value of something; the value of a cell in Microsoft Excel or the text of a selection in Microsoft Word. `service` is typically the name of an application, `topic` is typically the name of the file, and `item` is application-specific. The command returns the value of `item` as defined in the application. Normally this is interpreted to be a string with terminating null. If `-binary` is specified, the result is returned as a byte array.

**dde services** `service topic`  
**dde services** returns a list of service-topic pairs that currently exist on the machine. If `service` and `topic` are both empty strings (`{}`), then all service-topic pairs currently available on the system are returned. If `service` is empty and `topic` is not, then all services with the specified topic are returned. If `service` is non-empty and `topic` is, all topics for a given service are returned. If both are non-empty, if that service-topic pair currently exists, it is returned; otherwise, an empty string is returned.

**dde eval** `-async? topic cmd ?arg arg ...?`  
**dde eval** evaluates a command and its arguments using the interpreter specified by `topic`. The DDE service must be the **TclEval** service. The -async option requests asynchronous invocation. The command returns an error message if the script did not run, unless the -async flag was used, in which case the command returns immediately with no error. This command can be used to replace
DDE AND TCL

A Tcl interpreter always has a service name of TclEval. Each different interpreter of all running Tcl applications must be given a unique name specified by dde servername. Each interp is available as a DDE topic only if the dde servername command was used to set the name of the topic for each interp. So a dde services TclEval {} command will return a list of service-topic pairs, where each of the currently running interps will be a topic.

When Tcl processes a dde execute command, the data for the execute is run as a script in the interp named by the topic of the dde execute command.

When Tcl processes a dde request command, it returns the value of the variable given in the dde command in the context of the interp named by the dde topic. Tcl reserves the variable $TCLEVAL$EXECUTE$RESULT for internal use, and dde request commands for that variable will give unpredictable results.

An external application which wishes to run a script in Tcl should have that script store its result in a variable, run the dde execute command, and the run dde request to get the value of the variable.

When using DDE, be careful to ensure that the event queue is flushed using either update or vwait. This happens by default when using wish unless a blocking command is called (such as exec without adding the & to place the process in the background). If for any reason the event queue is not flushed, DDE commands may hang until the event queue is flushed. This can create a deadlock situation.

EXAMPLE

This asks Internet Explorer (which must already be running) to go to a particularly important website:
package require dde

dde execute iexplore WWW_OpenURL http://www.tcl.tk/

SEE ALSO

tk, winfo, send

KEYWORDS

application, dde, name, remote execution

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997 Sun Microsystems, Inc.
Copyright © 2001 ActiveState Corporation.
**NAME**

incr - Increment the value of a variable

**SYNOPSIS**

`incr varName ?increment?`

**DESCRIPTION**

Increments the value stored in the variable whose name is `varName`. The value of the variable must be an integer. If `increment` is supplied then its value (which must be an integer) is added to the value of variable `varName`; otherwise 1 is added to `varName`. The new value is stored as a decimal string in variable `varName` and also returned as result.

Starting with the Tcl 8.5 release, the variable `varName` passed to `incr` may be unset, and in that case, it will be set to the value `increment` or to the default increment value of 1.

**EXAMPLES**

Add one to the contents of the variable `x`:

```incr x```

Add 42 to the contents of the variable `x`:

```incr x 42```
Add the contents of the variable $y$ to the contents of the variable $x$:

```
incr x $y
```

Add nothing at all to the variable $x$ (often useful for checking whether an argument to a procedure is actually integral and generating an error if it is not):

```
incr x 0
```

**SEE ALSO**

*expr*

**KEYWORDS**

*add, increment, variable, value*
NAME

subst - Perform backslash, command, and variable substitutions

SYNOPSIS

subst ?-nobackslashes? ?-nocommands? ?-novariables? string

DESCRIPTION

This command performs variable substitutions, command substitutions, and backslash substitutions on its string argument and returns the fully-substituted result. The substitutions are performed in exactly the same way as for Tcl commands. As a result, the string argument is actually substituted twice, once by the Tcl parser in the usual fashion for Tcl commands, and again by the subst command.

If any of the -nobackslashes, -nocommands, or -novariables are specified, then the corresponding substitutions are not performed. For example, if -nocommands is specified, command substitution is not performed: open and close brackets are treated as ordinary characters with no special interpretation.

Note that the substitution of one kind can include substitution of other kinds. For example, even when the -novariables option is specified, command substitution is performed without restriction. This means that any variable substitution necessary to complete the command substitution will still take place. Likewise, any command substitution necessary to complete a variable substitution will take place, even when -nocommands is specified. See the EXAMPLES below.

If an error occurs during substitution, then subst will return that error. If
a break exception occurs during command or variable substitution, the result of the whole substitution will be the string (as substituted) up to the start of the substitution that raised the exception. If a continue exception occurs during the evaluation of a command or variable substitution, an empty string will be substituted for that entire command or variable substitution (as long as it is well-formed Tcl.) If a return exception occurs, or any other return code is returned during command or variable substitution, then the returned value is substituted for that substitution. See the EXAMPLES below. In this way, all exceptional return codes are “caught” by subst. The subst command itself will either return an error, or will complete successfully.

**EXAMPLES**

When it performs its substitutions, subst does not give any special treatment to double quotes or curly braces (except within command substitutions) so the script

```
set a 44
subst {xyz {a}}
```

returns “xyz {44}”, not “xyz ${a}” and the script

```
set a "p\} q \{r"
subst {xyz {a}}
```

returns “xyz {p} q {r}”, not “xyz {p\} q \{r}”.

When command substitution is performed, it includes any variable substitution necessary to evaluate the script.

```
set a 44
subst -novariables {a [format $a]}
```
returns “$a 44”, not “$a $a”. Similarly, when variable substitution is performed, it includes any command substitution necessary to retrieve the value of the variable.

```
proc b {} {return c}
array set a {c c [b] tricky}
subst -nocommands {[} $a([}]
```

returns “[b] c”, not “[b] tricky”.

The continue and break exceptions allow command substitutions to prevent substitution of the rest of the command substitution and the rest of string respectively, giving script authors more options when processing text using subst. For example, the script

```
subst {abc,[break],def}
```

returns “abc,”, not “abc,,def” and the script

```
subst {abc,[continue;expr {1+2}],def}
```

returns “abc,,def”, not “abc,3,def”.

Other exceptional return codes substitute the returned value

```
subst {abc,[return foo;expr {1+2}],def}
```

returns “abc,foo,def”, not “abc,3,def” and

```
subst {abc,[return -code 10 foo;expr {1+2}],def}
```
also returns “\texttt{abc,foo,def}”, not “\texttt{abc,3,def}”.

**SEE ALSO**

\texttt{Tcl, eval, break, continue}

**KEYWORDS**

backslash substitution, command substitution, variable substitution
NAME
dict - Manipulate dictionaries
SYNOPSIS
DESCRIPTION
dict append dictionaryVariable key ?string ...?
dict create ?key value ...?
dict exists dictionaryValue key ?key ...?
dict filter dictionaryValue filterType arg ?arg ...
dict filter dictionaryValue key globPattern
dict filter dictionaryValue script {keyVar valueVar} script
dict filter dictionaryValue value globPattern
dict for {keyVar valueVar} dictionaryValue body
dict get dictionaryValue ?key ...?
dict incr dictionaryVariable key ?increment?
dict info dictionaryValue
dict keys dictionaryValue ?globPattern?
dict lappend dictionaryVariable key ?value ...
dict merge ?dictionaryValue ...
dict remove dictionaryValue ?key ...
dict replace dictionaryValue ?key value ...
dict set dictionaryVariable key ?key ...? value
dict size dictionaryValue
dict unset dictionaryVariable key ?key ...
dict update dictionaryVariable key varName ?key varName ...
   body
dict values dictionaryValue ?globPattern?
dict with dictionaryVariable ?key ...? body
DICTIONARY VALUES
EXAMPLES
SEE ALSO
KEYWORDS
NAME

dict - Manipulate dictionaries

SYNOPSIS

dict option arg ?arg ...?

DESCRIPTION

Performs one of several operations on dictionary values or variables containing dictionary values (see the DICTIONARY VALUES section below for a description), depending on option. The legal options (which may be abbreviated) are:

**dict append** dictionaryVariable key ?string ...?
   This appends the given string (or strings) to the value that the given key maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable. Non-existent keys are treated as if they map to an empty string.

**dict create** ?key value ...?
   Create a new dictionary that contains each of the key/value mappings listed as arguments (keys and values alternating, with each key being followed by its associated value.)

**dict exists** dictionaryValue key ?key ...?
   This returns a boolean value indicating whether the given key (or path of keys through a set of nested dictionaries) exists in the given dictionary value. This returns a true value exactly when dict get on that path will succeed.

**dict filter** dictionaryValue filterType arg ?arg ...?
   This takes a dictionary value and returns a new dictionary that contains just those key/value pairs that match the specified filter type (which may be abbreviated.) Supported filter types are:

   **dict filter** dictionaryValue key globPattern
The key rule only matches those key/value pairs whose keys match the given pattern (in the style of \texttt{string match}).

\textbf{dict filter} \texttt{dictionaryValue script} \{keyVar valueVar\} script  \(\text{\texttt{script}}\)

The script rule tests for matching by assigning the key to the \texttt{keyVar} and the value to the \texttt{valueVar}, and then evaluating the given script which should return a boolean value (with the key/value pair only being included in the result of the \texttt{dict filter} when a true value is returned.) Note that the first argument after the rule selection word is a two-element list. If the \texttt{script} returns with a condition of \texttt{TCL\_BREAK}, no further key/value pairs are considered for inclusion in the resulting dictionary, and a condition of \texttt{TCL\_CONTINUE} is equivalent to a false result. The key/value pairs are tested in the order in which the keys were inserted into the dictionary.

\textbf{dict filter} \texttt{dictionaryValue value globPattern}  \(\text{\texttt{value}}\)

The value rule only matches those key/value pairs whose values match the given pattern (in the style of \texttt{string match}).

\textbf{dict for} \{keyVar valueVar\} \texttt{dictionaryValue body}  \(\text{\texttt{body}}\)

This command takes three arguments, the first a two-element list of variable names (for the key and value respectively of each mapping in the dictionary), the second the dictionary value to iterate across, and the third a script to be evaluated for each mapping with the key and value variables set appropriately (in the manner of \texttt{foreach}.) The result of the command is an empty string. If any evaluation of the body generates a \texttt{TCL\_BREAK} result, no further pairs from the dictionary will be iterated over and the \texttt{dict for} command will terminate successfully immediately. If any evaluation of the body generates a \texttt{TCL\_CONTINUE} result, this shall be treated exactly like a normal \texttt{TCL\_OK} result. The order of iteration is the order in which the keys were inserted into the dictionary.

\textbf{dict get} \texttt{dictionaryValue} ?key \ldots?  \(\text{\texttt{?key \ldots?}}\)

Given a dictionary value (first argument) and a key (second argument), this will retrieve the value for that key. Where several keys are supplied, the behaviour of the command shall be as if the
result of `dict get $dictVal $key` was passed as the first argument to `dict get` with the remaining arguments as second (and possibly subsequent) arguments. This facilitates lookups in nested dictionaries. For example, the following two commands are equivalent:

```
dict get $dict foo bar spong
dict get [dict get [dict get $dict foo] bar] spong
```

If no keys are provided, `dict` would return a list containing pairs of elements in a manner similar to `array get`. That is, the first element of each pair would be the key and the second element would be the value for that key. It is an error to attempt to retrieve a value for a key that is not present in the dictionary.

**dict incr dictionaryVariable key ?increment?**
This adds the given increment value (an integer that defaults to 1 if not specified) to the value that the given key maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable. Non-existent keys are treated as if they map to 0. It is an error to increment a value for an existing key if that value is not an integer.

**dict info dictionaryValue**
This returns information (intended for display to people) about the given dictionary though the format of this data is dependent on the implementation of the dictionary. For dictionaries that are implemented by hash tables, it is expected that this will return the string produced by `Tcl_HashStats`, similar to `array info`.

**dict keys dictionaryValue ?globPattern?**
Return a list of all keys in the given dictionary value. If a pattern is supplied, only those keys that match it (according to the rules of `string match`) will be returned. The returned keys will be in the order that they were inserted into the dictionary.
dict lappend dictionaryVariable key ?value ...?
This appends the given items to the list value that the given key maps to in the dictionary value contained in the given variable, writing the resulting dictionary value back to that variable. Non-existent keys are treated as if they map to an empty list, and it is legal for there to be no items to append to the list. It is an error for the value that the key maps to to not be representable as a list.

dict merge ?dictionaryValue ...?
Return a dictionary that contains the contents of each of the dictionaryValue arguments. Where two (or more) dictionaries contain a mapping for the same key, the resulting dictionary maps that key to the value according to the last dictionary on the command line containing a mapping for that key.

dict remove dictionaryValue ?key ...?
Return a new dictionary that is a copy of an old one passed in as first argument except without mappings for each of the keys listed. It is legal for there to be no keys to remove, and it also legal for any of the keys to be removed to not be present in the input dictionary in the first place.

dict replace dictionaryValue ?key value ...?
Return a new dictionary that is a copy of an old one passed in as first argument except with some values different or some extra key/value pairs added. It is legal for this command to be called with no key/value pairs, but illegal for this command to be called with a key but no value.

dict set dictionaryVariable key ?key ...? value
This operation takes the name of a variable containing a dictionary value and places an updated dictionary value in that variable containing a mapping from the given key to the given value. When multiple keys are present, this operation creates or updates a chain of nested dictionaries.

dict size dictionaryValue
Return the number of key/value mappings in the given dictionary
value.

**dict unset** *dictionaryVariable key ?key ...?*
This operation (the companion to **dict set**) takes the name of a variable containing a dictionary value and places an updated dictionary value in that variable that does not contain a mapping for the given key. Where multiple keys are present, this describes a path through nested dictionaries to the mapping to remove. At least one key must be specified, but the last key on the key-path need not exist. All other components on the path must exist.

**dict update** *dictionaryVariable key varName ?key varName ...? body*
Execute the Tcl script in *body* with the value for each *key* (as found by reading the dictionary value in *dictionaryVariable*) mapped to the variable *varName*. There may be multiple *key/varName* pairs. If a *key* does not have a mapping, that corresponds to an unset *varName*. When *body* terminates, any changes made to the *varNames* is reflected back to the dictionary within *dictionaryVariable* (unless *dictionaryVariable* itself becomes unreadable, when all updates are silently discarded), even if the result of *body* is an error or some other kind of exceptional exit. The result of **dict update** is (unless some kind of error occurs) the result of the evaluation of *body*. Note that the mapping of values to variables does not use traces; changes to the *dictionaryVariable*'s contents only happen when *body* terminates.

**dict values** *dictionaryValue ?globPattern?*
Return a list of all values in the given dictionary value. If a pattern is supplied, only those values that match it (according to the rules of **string match**) will be returned. The returned values will be in the order of that the keys associated with those values were inserted into the dictionary.

**dict with** *dictionaryVariable ?key ...? body*
Execute the Tcl script in *body* with the value for each key in *dictionaryVariable* mapped (in a manner similarly to **dict update**) to a variable with the same name. Where one or more *keys* are available, these indicate a chain of nested dictionaries, with the
innermost dictionary being the one opened out for the execution of body. As with dict update, making dictionaryVariable unreadable will make the updates to the dictionary be discarded, and this also happens if the contents of dictionaryVariable are adjusted so that the chain of dictionaries no longer exists. The result of dict with is (unless some kind of error occurs) the result of the evaluation of body. Note that the mapping of values to variables does not use traces; changes to the dictionaryVariable's contents only happen when body terminates.

**DICTIONARY VALUES**

Dictionaries are values that contain an efficient, order-preserving mapping from arbitrary keys to arbitrary values. Each key in the dictionary maps to a single value. They have a textual format that is exactly that of any list with an even number of elements, with each mapping in the dictionary being represented as two items in the list. When a command takes a dictionary and produces a new dictionary based on it (either returning it or writing it back into the variable that the starting dictionary was read from) the new dictionary will have the same order of keys, modulo any deleted keys and with new keys added on to the end. When a string is interpreted as a dictionary and it would otherwise have duplicate keys, only the last value for a particular key is used; the others are ignored, meaning that, “apple banana” and “apple carrot apple banana” are equivalent dictionaries (with different string representations).

**EXAMPLES**

Constructing and using nested dictionaries:

```plaintext
# Data for one employee
dict set employeeInfo 12345-A forenames "Joe"
dict set employeeInfo 12345-A surname "Schmoe"
dict set employeeInfo 12345-A street "147 Short Street"
dict set employeeInfo 12345-A city "Springfield"
dict set employeeInfo 12345-A phone "555-1234"
```
# Data for another employee

```
dict set employeeInfo 98372-J forenames "Anne"
dict set employeeInfo 98372-J surname "Other"
dict set employeeInfo 98372-J street "32995 Oakdale"
dict set employeeInfo 98372-J city "Springfield"
dict set employeeInfo 98372-J phone "555-8765"
```

# The above data probably ought to come from a database...

# Print out some employee info

```
set i 0
puts "There are [dict size $employeeInfo] employees"
dict for {id info} $employeeInfo {
  puts "Employee #*[incr i]: $id"
  dict with info {
    puts " Name: $forenames $surname"
    puts " Address: $street, $city"
    puts " Telephone: $phone"
  }
}
```

# Another way to iterate and pick out names...

```
foreach id [dict keys $employeeInfo] {
  puts "Hello, [dict get $employeeInfo $id forename"
}
```

A localizable version of `string toupper`:

```
# Set up the basic C locale
set capital [dict create C [dict create]]
foreach c [split {abcdefghijklmnopqrstuvwxyz} ""] {
  dict set capital C $c [string toupper $c]
}
```

# English locales can luckily share the "C" locale
```
dict set capital en [dict get $capital C]
dict set capital en_US [dict get $capital C]
```
**SEE ALSO**

append, array, foreach, incr, list, lappend, set

**KEYWORDS**

dictionary, create, update, lookup, iterate, filter
NAME
info - Return information about the state of the Tcl interpreter

SYNOPSIS

DESCRIPTION

info args procname
info body procname
info cmdcount
info commands ?pattern?
info complete command
info default procname arg varname
info exists varName
info frame ?number?
  type
    source
    proc
eval
  precompiled
line
file
cmd
proc
lambda
level
info functions ?pattern?
info globals ?pattern?
info hostname
info level ?number?
info library
info loaded ?interp?
info locals ?pattern?
info nameofexecutable
### NAME

info - Return information about the state of the Tcl interpreter

### SYNOPSIS

```bash
info option ?arg arg ...?  
```

### DESCRIPTION

This command provides information about various internals of the Tcl interpreter. The legal `options` (which may be abbreviated) are:

**info args procname**

Returns a list containing the names of the arguments to procedure `procname`, in order. `Procname` must be the name of a Tcl command procedure.

**info body procname**

Returns the body of procedure `procname`. `Procname` must be the name of a Tcl command procedure.

**info cmdcount**

Returns a count of the total number of commands that have been invoked in this interpreter.

**info commands ?pattern?**

If `pattern` is not specified, returns a list of names of all the Tcl
commands visible (i.e. executable without using a qualified name) to the current namespace, including both the built-in commands written in C and the command procedures defined using the `proc` command. If `pattern` is specified, only those names matching `pattern` are returned. Matching is determined using the same rules as for `string match`. `pattern` can be a qualified name like `Foo::print*`. That is, it may specify a particular namespace using a sequence of namespace names separated by double colons (::), and may have pattern matching special characters at the end to specify a set of commands in that namespace. If `pattern` is a qualified name, the resulting list of command names has each one qualified with the name of the specified namespace, and only the commands defined in the named namespace are returned.

**info complete** `command`

Returns 1 if `command` is a complete Tcl command in the sense of having no unclosed quotes, braces, brackets or array element names. If the command does not appear to be complete then 0 is returned. This command is typically used in line-oriented input environments to allow users to type in commands that span multiple lines; if the command is not complete, the script can delay evaluating it until additional lines have been typed to complete the command.

**info default** `procname arg varname`

`Procname` must be the name of a Tcl command procedure and `arg` must be the name of an argument to that procedure. If `arg` does not have a default value then the command returns 0. Otherwise it returns 1 and places the default value of `arg` into variable `varname`.

**info exists** `varName`

Returns 1 if the variable named `varName` exists in the current context (either as a global or local variable) and has been defined by being given a value, returns 0 otherwise.

**info frame** ?`number`?

This command provides access to all frames on the stack, even those hidden from `info level`. If `number` is not specified, this
command returns a number giving the frame level of the command. This is 1 if the command is invoked at top-level. If number is specified, then the result is a dictionary containing the location information for the command at the numbered level on the stack.

If number is positive (> 0) then it selects a particular stack level (1 refers to the top-most active command, i.e., info frame itself, 2 to the command it was called from, and so on); otherwise it gives a level relative to the current command (0 refers to the current command, i.e., info frame itself, -1 to its caller, and so on).

This is similar to how info level works, except that this subcommand reports all frames, like sourced scripts, evals, uplevels, etc.

Note that for nested commands, like “foo [bar [x]]”, only “x” will be seen by an info frame invoked within “x”. This is the same as for info level and error stack traces.

The result dictionary may contain the keys listed below, with the specified meanings for their values:

**type**

This entry is always present and describes the nature of the location for the command. The recognized values are source, proc, eval, and precompiled.

**source**

means that the command is found in a script loaded by the source command.

**proc**

means that the command is found in dynamically created procedure body.

**eval**

means that the command is executed by eval or uplevel.

**precompiled**
means that the command is found in a precompiled script (loadable by the package \texttt{tbcload}), and no further information will be available.

**line**
This entry provides the number of the line the command is at inside of the script it is a part of. This information is not present for type \texttt{precompiled}. For type \texttt{source} this information is counted relative to the beginning of the file, whereas for the last two types the line is counted relative to the start of the script.

**file**
This entry is present only for type \texttt{source}. It provides the normalized path of the file the command is in.

**cmd**
This entry provides the string representation of the command. This is usually the unsubstituted form, however for commands which are a pure list executed by eval it is the substituted form as they have no other string representation. Care is taken that the pure-List property of the latter is not spoiled.

**proc**
This entry is present only if the command is found in the body of a regular Tcl procedure. It then provides the name of that procedure.

**lambda**
This entry is present only if the command is found in the body of an anonymous Tcl procedure, i.e. a lambda. It then provides the entire definition of the lambda in question.

**level**
This entry is present only if the queried frame has a corresponding frame returned by \texttt{info level}. It provides the index of this frame, relative to the current level (0 and negative numbers).
A thing of note is that for procedures statically defined in files the locations of commands in their bodies will be reported with type `source` and absolute line numbers, and not as type `proc`. The same is true for procedures nested in statically defined procedures, and literal eval scripts in files or statically defined procedures.

In contrast, a procedure definition or `eval` within a dynamically `evaluated` environment count linenumbers relative to the start of their script, even if they would be able to count relative to the start of the outer dynamic script. That type of number usually makes more sense.

A different way of describing this behaviour is that file based locations are tracked as deeply as possible, and where this is not possible the lines are counted based on the smallest possible `eval` or procedure body, as that scope is usually easier to find than any dynamic outer scope.

The syntactic form `{*}` is handled like `eval`. I.e. if it is given a literal list argument the system tracks the linenumber within the list words as well, and otherwise all linenumbers are counted relative to the start of each word (smallest scope)

**info functions ?pattern?**
If `pattern` is not specified, returns a list of all the math functions currently defined. If `pattern` is specified, only those functions whose name matches `pattern` are returned. Matching is determined using the same rules as for `string match`.

**info globals ?pattern?**
If `pattern` is not specified, returns a list of all the names of currently-defined global variables. Global variables are variables in the global namespace. If `pattern` is specified, only those names matching `pattern` are returned. Matching is determined using the same rules as for `string match`.

**info hostname**
Returns the name of the computer on which this invocation is being
executed. Note that this name is not guaranteed to be the fully qualified domain name of the host. Where machines have several different names (as is common on systems with both TCP/IP (DNS) and NetBIOS-based networking installed,) it is the name that is suitable for TCP/IP networking that is returned.

**info level ?*number***?

If *number* is not specified, this command returns a number giving the stack level of the invoking procedure, or 0 if the command is invoked at top-level. If *number* is specified, then the result is a list consisting of the name and arguments for the procedure call at level *number* on the stack. If *number* is positive then it selects a particular stack level (1 refers to the top-most active procedure, 2 to the procedure it called, and so on); otherwise it gives a level relative to the current level (0 refers to the current procedure, -1 to its caller, and so on). See the **uplevel** command for more information on what stack levels mean.

**info library**

Returns the name of the library directory in which standard Tcl scripts are stored. This is actually the value of the **tcl_library** variable and may be changed by setting **tcl_library**. See the **tclvars** manual entry for more information.

**info loaded ?*interp***?

Returns a list describing all of the packages that have been loaded into *interp* with the **load** command. Each list element is a sub-list with two elements consisting of the name of the file from which the package was loaded and the name of the package. For statically-loaded packages the file name will be an empty string. If *interp* is omitted then information is returned for all packages loaded in any interpreter in the process. To get a list of just the packages in the current interpreter, specify an empty string for the *interp* argument.

**info locals ?*pattern***?

If *pattern* is not specified, returns a list of all the names of currently-defined local variables, including arguments to the current procedure, if any. Variables defined with the **global**, **upvar** and
variable commands will not be returned. If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for string match.

info nameofexecutable
Returns the full path name of the binary file from which the application was invoked. If Tcl was unable to identify the file, then an empty string is returned.

info patchlevel
Returns the value of the global variable tcl_patchLevel; see the tclvars manual entry for more information.

info procs ?pattern?
If pattern is not specified, returns a list of all the names of Tcl command procedures in the current namespace. If pattern is specified, only those procedure names in the current namespace matching pattern are returned. Matching is determined using the same rules as for string match. If pattern contains any namespace separators, they are used to select a namespace relative to the current namespace (or relative to the global namespace if pattern starts with ::) to match within; the matching pattern is taken to be the part after the last namespace separator.

info script ?filename?
If a Tcl script file is currently being evaluated (i.e. there is a call to Tcl_EvalFile active or there is an active invocation of the source command), then this command returns the name of the innermost file being processed. If filename is specified, then the return value of this command will be modified for the duration of the active invocation to return that name. This is useful in virtual file system applications. Otherwise the command returns an empty string.

info sharedlibextension
Returns the extension used on this platform for the names of files containing shared libraries (for example, .so under Solaris). If shared libraries are not supported on this platform then an empty string is returned.
info tclversion
Returns the value of the global variable tcl_version; see the
tclvars manual entry for more information.

info vars ?pattern?  
If pattern is not specified, returns a list of all the names of currently-visible variables. This includes locals and currently-visible globals. If pattern is specified, only those names matching pattern are returned. Matching is determined using the same rules as for
string match. pattern can be a qualified name like Foo::option*. That is, it may specify a particular namespace using a sequence of namespace names separated by double colons (::), and may have pattern matching special characters at the end to specify a set of variables in that namespace. If pattern is a qualified name, the resulting list of variable names has each matching namespace variable qualified with the name of its namespace. Note that a currently-visible variable may not yet “exist” if it has not been set (e.g. a variable declared but not set by variable).

**EXAMPLE**

This command prints out a procedure suitable for saving in a Tcl script:

```tcl
proc printProc {procName} {
    set result [list proc $procName]
    set formals {}
    foreach var [info args $procName] {
        if {[info default $procName $var def]} {
            lappend formals [list $var $def]
        } else {
            # Still need the list-quoting because variable names may properly contain spaces.
            lappend formals [list $var]
        }
    }
    puts [lappend result $formals [info body $procName]]
}
```
SEE ALSO

global, proc

KEYWORDS

command, information, interpreter, level, namespace, procedure, variable

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
Copyright © 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright © 1998-2000 Ajuba Solutions
NAME

pid - Retrieve process identifiers

SYNOPSIS

pid ?fileId?

DESCRIPTION

If the fileId argument is given then it should normally refer to a process pipeline created with the open command. In this case the pid command will return a list whose elements are the process identifiers of all the processes in the pipeline, in order. The list will be empty if fileId refers to an open file that is not a process pipeline. If no fileId argument is given then pid returns the process identifier of the current process. All process identifiers are returned as decimal strings.

EXAMPLE

Print process information about the processes in a pipeline using the SysV ps program before reading the output of that pipeline:

```tcl
set pipeline [open "| zcat somefile.gz | grep foobar"
# Print process information
exec ps -fp [pid $pipeline] @stdout
# Print a separator and then the output of the pipeline
puts [string repeat - 70]
puts [read $pipeline]
close $pipeline
```
SEE ALSO

exec, open

KEYWORDS

file, pipeline, process identifier
NAME

switch - Evaluate one of several scripts, depending on a given value

SYNOPSIS

DESCRIPTION

-exact
-glob
-regexp
-nocase
-matchvar varName
-indexvar varName
--

EXAMPLES

SEE ALSO

KEYWORDS

NAME

switch - Evaluate one of several scripts, depending on a given value

SYNOPSIS

switch ?options? string pattern body ?pattern body ...?
switch ?options? string {pattern body ?pattern body ...?}

DESCRIPTION

The switch command matches its string argument against each of the pattern arguments in order. As soon as it finds a pattern that matches string it evaluates the following body argument by passing it recursively to the Tcl interpreter and returns the result of that evaluation. If the last pattern argument is default then it matches anything. If no pattern
argument matches *string* and no default is given, then the **switch** command returns an empty string.

If the initial arguments to **switch** start with - then they are treated as options unless there are exactly two arguments to **switch** (in which case the first must be the *string* and the second must be the *pattern/body* list). The following options are currently supported:

- **exact**
  Use exact matching when comparing *string* to a pattern. This is the default.

- **glob**
  When matching *string* to the patterns, use glob-style matching (i.e. the same as implemented by the **string match** command).

- **regexp**
  When matching *string* to the patterns, use regular expression matching (as described in the **re_syntax** reference page).

- **nocase**
  Causes comparisons to be handled in a case-insensitive manner.

- **matchvar** `varName`
  This option (only legal when **regexp** is also specified) specifies the name of a variable into which the list of matches found by the regular expression engine will be written. The first element of the list written will be the overall substring of the input string (i.e. the *string* argument to **switch**) matched, the second element of the list will be the substring matched by the first capturing parenthesis in the regular expression that matched, and so on. When a **default** branch is taken, the variable will have the empty list written to it. This option may be specified at the same time as the **indexvar** option.

- **indexvar** `varName`
  This option (only legal when **regexp** is also specified) specifies the name of a variable into which the list of indices referring to
matching substrings found by the regular expression engine will be written. The first element of the list written will be a two-element list specifying the index of the start and index of the first character after the end of the overall substring of the input string (i.e. the \texttt{string} argument to \texttt{switch}) matched, in a similar way to the \texttt{-indices} option to the \texttt{regexp} can obtain. Similarly, the second element of the list refers to the first capturing parenthesis in the regular expression that matched, and so on. When a \texttt{default} branch is taken, the variable will have the empty list written to it. This option may be specified at the same time as the \texttt{-matchvar} option.

\texttt{--}

Marks the end of options. The argument following this one will be treated as \texttt{string} even if it starts with a \texttt{-}. This is not required when the matching patterns and bodies are grouped together in a single argument.

Two syntaxes are provided for the \texttt{pattern} and \texttt{body} arguments. The first uses a separate argument for each of the patterns and commands; this form is convenient if substitutions are desired on some of the patterns or commands. The second form places all of the patterns and commands together into a single argument; the argument must have proper list structure, with the elements of the list being the patterns and commands. The second form makes it easy to construct multi-line \texttt{switch} commands, since the braces around the whole list make it unnecessary to include a backslash at the end of each line. Since the \texttt{pattern} arguments are in braces in the second form, no command or variable substitutions are performed on them; this makes the behavior of the second form different than the first form in some cases.

If a \texttt{body} is specified as "-" it means that the \texttt{body} for the next pattern should also be used as the body for this pattern (if the next pattern also has a body of "-" then the body after that is used, and so on). This feature makes it possible to share a single \texttt{body} among several patterns.

Beware of how you place comments in \texttt{switch} commands. Comments should only be placed \texttt{inside} the execution body of one of the patterns,
and not intermingled with the patterns.

**EXAMPLES**

The `switch` command can match against variables and not just literals, as shown here (the result is 2):

```plaintext
set foo "abc"
switch abc a - b {expr {1}} $foo {expr {2}} default {expr {3}}
```

Using glob matching and the fall-through body is an alternative to writing regular expressions with alternations, as can be seen here (this returns 1):

```plaintext
switch -glob aaab {
  a*b
  b {expr {1}}
  a* {expr {2}}
  default {expr {3}}
}
```

Whenever nothing matches, the `default` clause (which must be last) is taken. This example has a result of 3:

```plaintext
switch xyz {
  a -
  b {
      # Correct Comment Placement
      expr {1}
  }
  c {
      expr {2}
  }
```
When matching against regular expressions, information about what exactly matched is easily obtained using the `-matchvar` option:

```
switch -regexp -matchvar foo -- $bar {
  a(b*)c {
    puts "Found [string length [lindex $foo 1]] 'b"
  }
  d(e*)f(g*)h {
    puts "Found [string length [lindex $foo 1]] 'e"
    [string length [lindex $foo 2]] 'g's"
  }
}
```

SEE ALSO

`for`, `if`, `regexp`

KEYWORDS

`switch`, `match`, `regular expression`
NAME
encoding - Manipulate encodings

SYNOPSIS
encoding option ?arg arg ...?

INTRODUCTION
Strings in Tcl are encoded using 16-bit Unicode characters. Different operating system interfaces or applications may generate strings in other encodings such as Shift-JIS. The encoding command helps to bridge the gap between Unicode and these other formats.

DESCRIPTION
Performs one of several encoding related operations, depending on
option. The legal options are:

**encoding convertfrom ?encoding? data**
Convert *data* to Unicode from the specified *encoding*. The characters in *data* are treated as binary data where the lower 8-bits of each character is taken as a single byte. The resulting sequence of bytes is treated as a string in the specified *encoding*. If *encoding* is not specified, the current system encoding is used.

**encoding convertto ?encoding? string**
Convert *string* from Unicode to the specified *encoding*. The result is a sequence of bytes that represents the converted string. Each byte is stored in the lower 8-bits of a Unicode character. If *encoding* is not specified, the current system encoding is used.

**encoding dirs ?directoryList?**
Tcl can load encoding data files from the file system that describe additional encodings for it to work with. This command sets the search path for *.enc* encoding data files to the list of directories *directoryList*. If *directoryList* is omitted then the command returns the current list of directories that make up the search path. It is an error for *directoryList* to not be a valid list. If, when a search for an encoding data file is happening, an element in *directoryList* does not refer to a readable, searchable directory, that element is ignored.

**encoding names**
Returns a list containing the names of all of the encodings that are currently available.

**encoding system ?encoding?**
Set the system encoding to *encoding*. If *encoding* is omitted then the command returns the current system encoding. The system encoding is used whenever Tcl passes strings to system calls.

**EXAMPLE**

It is common practice to write script files using a text editor that
produces output in the euc-jp encoding, which represents the ASCII characters as single bytes and Japanese characters as two bytes. This makes it easy to embed literal strings that correspond to non-ASCII characters by simply typing the strings in place in the script. However, because the `source` command always reads files using the current system encoding, Tcl will only source such files correctly when the encoding used to write the file is the same. This tends not to be true in an internationalized setting. For example, if such a file was sourced in North America (where the ISO8859-1 is normally used), each byte in the file would be treated as a separate character that maps to the 00 page in Unicode. The resulting Tcl strings will not contain the expected Japanese characters. Instead, they will contain a sequence of Latin-1 characters that correspond to the bytes of the original string. The `encoding` command can be used to convert this string to the expected Japanese Unicode characters. For example,

```
set s [encoding convertfrom euc-jp "\xA4\xCF"]
```

would return the Unicode string “\u306F”, which is the Hiragana letter HA.

SEE ALSO

Tcl_GetEncoding

KEYWORDS

encoding

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1998 by Scriptics Corporation.
NAME
 interp - Create and manipulate Tcl interpreters

SYNOPSIS

DESCRIPTION

THE interp COMMAND

text

SLAVE COMMAND

text
**NAME**

interp - Create and manipulate Tcl interpreters

**SYNOPSIS**

```bash
 interp subcommand ?arg arg ...?
```

**DESCRIPTION**
This command makes it possible to create one or more new Tcl interpreters that co-exist with the creating interpreter in the same application. The creating interpreter is called the master and the new interpreter is called a slave. A master can create any number of slaves, and each slave can itself create additional slaves for which it is master, resulting in a hierarchy of interpreters.

Each interpreter is independent from the others: it has its own name space for commands, procedures, and global variables. A master interpreter may create connections between its slaves and itself using a mechanism called an alias. An alias is a command in a slave interpreter which, when invoked, causes a command to be invoked in its master interpreter or in another slave interpreter. The only other connections between interpreters are through environment variables (the env variable), which are normally shared among all interpreters in the application, and by resource limit exceeded callbacks. Note that the name space for files (such as the names returned by the open command) is no longer shared between interpreters. Explicit commands are provided to share files and to transfer references to open files from one interpreter to another.

The interp command also provides support for safe interpreters. A safe interpreter is a slave whose functions have been greatly restricted, so that it is safe to execute untrusted scripts without fear of them damaging other interpreters or the application's environment. For example, all IO channel creation commands and subprocess creation commands are made inaccessible to safe interpreters. See SAFE INTERPRETERS below for more information on what features are present in a safe interpreter. The dangerous functionality is not removed from the safe interpreter; instead, it is hidden, so that only trusted interpreters can obtain access to it. For a detailed explanation of hidden commands, see HIDDEN COMMANDS, below. The alias mechanism can be used for protected communication (analogous to a kernel call) between a slave interpreter and its master. See ALIAS INVOCATION, below, for more details on how the alias mechanism works.

A qualified interpreter name is a proper Tcl lists containing a subset of its ancestors in the interpreter hierarchy, terminated by the string
naming the interpreter in its immediate master. Interpreter names are relative to the interpreter in which they are used. For example, if a is a slave of the current interpreter and it has a slave a1, which in turn has a slave a11, the qualified name of a11 in a is the list a1 a11.

The `interp` command, described below, accepts qualified interpreter names as arguments; the interpreter in which the command is being evaluated can always be referred to as `{}` (the empty list or string). Note that it is impossible to refer to a master (ancestor) interpreter by name in a slave interpreter except through aliases. Also, there is no global name by which one can refer to the first interpreter created in an application. Both restrictions are motivated by safety concerns.

## THE INTERP COMMAND

The `interp` command is used to create, delete, and manipulate slave interpreters, and to share or transfer channels between interpreters. It can have any of several forms, depending on the `subcommand` argument:

**interp alias srcPath srcToken**

Returns a Tcl list whose elements are the `targetCmd` and `args` associated with the alias represented by `srcToken` (this is the value returned when the alias was created; it is possible that the name of the source command in the slave is different from `srcToken`).

**interp alias srcPath srcToken `{}`**

Deletes the alias for `srcToken` in the slave interpreter identified by `srcPath`. `srcToken` refers to the value returned when the alias was created; if the source command has been renamed, the renamed command will be deleted.

**interp alias srcPath srcCmd targetPath targetCmd ?arg arg ...?**

This command creates an alias between one slave and another (see the `alias slave` command below for creating aliases between a slave and its master). In this command, either of the slave interpreters may be anywhere in the hierarchy of interpreters under the interpreter invoking the command. `SrcPath` and `srcCmd` identify
the source of the alias. *SrcPath* is a Tcl list whose elements select a particular interpreter. For example, “a b” identifies an interpreter b, which is a slave of interpreter a, which is a slave of the invoking interpreter. An empty list specifies the interpreter invoking the command. *srcCmd* gives the name of a new command, which will be created in the source interpreter. *TargetPath* and *targetCmd* specify a target interpreter and command, and the *arg* arguments, if any, specify additional arguments to *targetCmd* which are prepended to any arguments specified in the invocation of *srcCmd*. *TargetCmd* may be undefined at the time of this call, or it may already exist; it is not created by this command. The alias arranges for the given target command to be invoked in the target interpreter whenever the given source command is invoked in the source interpreter. See **ALIAS INVOCATION** below for more details. The command returns a token that uniquely identifies the command created *srcCmd*, even if the command is renamed afterwards. The token may but does not have to be equal to *srcCmd*.

**interp aliases ?path?**
This command returns a Tcl list of the tokens of all the source commands for aliases defined in the interpreter identified by *path*. The tokens correspond to the values returned when the aliases were created (which may not be the same as the current names of the commands).

**interp bgerror path ?cmdPrefix?**
This command either gets or sets the current background error handler for the interpreter identified by *path*. If *cmdPrefix* is absent, the current background error handler is returned, and if it is present, it is a list of words (of minimum length one) that describes what to set the interpreter's background error to. See the **BACKGROUND ERROR HANDLING** section for more details.

**interp create ?-safe? ?--? ?path?**
Creates a slave interpreter identified by *path* and a new command, called a *slave command*. The name of the slave command is the last component of *path*. The new slave interpreter and the slave command are created in the interpreter identified by the path
obtained by removing the last component from path. For example, if path is a b c then a new slave interpreter and slave command named c are created in the interpreter identified by the path a b.

The slave command may be used to manipulate the new interpreter as described below. If path is omitted, Tcl creates a unique name of the form interp$x, where x is an integer, and uses it for the interpreter and the slave command. If the -safe switch is specified (or if the master interpreter is a safe interpreter), the new slave interpreter will be created as a safe interpreter with limited functionality; otherwise the slave will include the full set of Tcl built-in commands and variables. The -- switch can be used to mark the end of switches; it may be needed if path is an unusual value such as -safe. The result of the command is the name of the new interpreter. The name of a slave interpreter must be unique among all the slaves for its master; an error occurs if a slave interpreter by the given name already exists in this master. The initial recursion limit of the slave interpreter is set to the current recursion limit of its parent interpreter.

**interp delete ?path ...?**

Deletes zero or more interpreters given by the optional path arguments, and for each interpreter, it also deletes its slaves. The command also deletes the slave command for each interpreter deleted. For each path argument, if no interpreter by that name exists, the command raises an error.

**interp eval path arg ?arg ...?**

This command concatenates all of the arg arguments in the same fashion as the concat command, then evaluates the resulting string as a Tcl script in the slave interpreter identified by path. The result of this evaluation (including all return options, such as -errorinfo and -errorcode information, if an error occurs) is returned to the invoking interpreter. Note that the script will be executed in the current context stack frame of the path interpreter; this is so that the implementations (in a master interpreter) of aliases in a slave interpreter can execute scripts in the slave that find out information about the slave's current state and stack frame.
interp exists path
Returns 1 if a slave interpreter by the specified path exists in this master, 0 otherwise. If path is omitted, the invoking interpreter is used.

interp expose path hiddenName ?exposedCmdName?
Makes the hidden command hiddenName exposed, eventually bringing it back under a new exposedCmdName name (this name is currently accepted only if it is a valid global name space name without any ::), in the interpreter denoted by path. If an exposed command with the targeted name already exists, this command fails. Hidden commands are explained in more detail in HIDDEN COMMANDS, below.

interp hide path exposedCmdName ?hiddenCmdName?
Makes the exposed command exposedCmdName hidden, renaming it to the hidden command hiddenCmdName, or keeping the same name if hiddenCmdName is not given, in the interpreter denoted by path. If a hidden command with the targeted name already exists, this command fails. Currently both exposedCmdName and hiddenCmdName can not contain namespace qualifiers, or an error is raised. Commands to be hidden by interp hide are looked up in the global namespace even if the current namespace is not the global one. This prevents slaves from fooling a master interpreter into hiding the wrong command, by making the current namespace be different from the global one. Hidden commands are explained in more detail in HIDDEN COMMANDS, below.

interp hidden path
Returns a list of the names of all hidden commands in the interpreter identified by path.

interp invokehidden path ?-option ...? hiddenCmdName ?arg ...?
Invokes the hidden command hiddenCmdName with the arguments supplied in the interpreter denoted by path. No substitutions or evaluation are applied to the arguments. Three -options are supported, all of which start with -: -namespace (which takes a
single argument afterwards, \textit{nsName}), \texttt{-global}, and \texttt{--}. If the \texttt{-namespace} flag is present, the hidden command is invoked in the namespace called \textit{nsName} in the target interpreter. If the \texttt{-global} flag is present, the hidden command is invoked at the global level in the target interpreter; otherwise it is invoked at the current call frame and can access local variables in that and outer call frames. The \texttt{--} flag allows the \textit{hiddenCmdName} argument to start with a “-” character, and is otherwise unnecessary. If both the \texttt{-namespace} and \texttt{-global} flags are present, the \texttt{-namespace} flag is ignored. Note that the hidden command will be executed (by default) in the current context stack frame of the \textit{path} interpreter. Hidden commands are explained in more detail in \textbf{HIDDEN COMMANDS}, below.

\textbf{interp limit} \textit{path} \texttt{limitType} \texttt{-option? \texttt{value} ...?}
Sets up, manipulates and queries the configuration of the resource limit \textit{limitType} for the interpreter denoted by \textit{path}. If no \texttt{-option} is specified, return the current configuration of the limit. If \texttt{-option} is the sole argument, return the value of that option. Otherwise, a list of \texttt{-option/value} argument pairs must supplied. See \textbf{RESOURCE LIMITS} below for a more detailed explanation of what limits and options are supported.

\textbf{interp issafe} \texttt{?path?}
Returns \texttt{1} if the interpreter identified by the specified \textit{path} is safe, \texttt{0} otherwise.

\textbf{interp marktrusted} \textit{path}
Marks the interpreter identified by \textit{path} as trusted. Does not expose the hidden commands. This command can only be invoked from a trusted interpreter. The command has no effect if the interpreter identified by \textit{path} is already trusted.

\textbf{interp recursionlimit} \textit{path} \texttt{?newlimit?}
Returns the maximum allowable nesting depth for the interpreter specified by \textit{path}. If \texttt{newlimit} is specified, the interpreter recursion limit will be set so that nesting of more than \texttt{newlimit} calls to \texttt{Tcl_Eval()} and related procedures in that interpreter will return an
error. The newlimit value is also returned. The newlimit value must be a positive integer between 1 and the maximum value of a non-long integer on the platform.

The command sets the maximum size of the Tcl call stack only. It cannot by itself prevent stack overflows on the C stack being used by the application. If your machine has a limit on the size of the C stack, you may get stack overflows before reaching the limit set by the command. If this happens, see if there is a mechanism in your system for increasing the maximum size of the C stack.

**interp share srcPath channelId destPath**
Causes the IO channel identified by channelId to become shared between the interpreter identified by srcPath and the interpreter identified by destPath. Both interpreters have the same permissions on the IO channel. Both interpreters must close it to close the underlying IO channel; IO channels accessible in an interpreter are automatically closed when an interpreter is destroyed.

**interp slaves ?path?**
Returns a Tcl list of the names of all the slave interpreters associated with the interpreter identified by path. If path is omitted, the invoking interpreter is used.

**interp target path alias**
Returns a Tcl list describing the target interpreter for an alias. The alias is specified with an interpreter path and source command name, just as in **interp alias** above. The name of the target interpreter is returned as an interpreter path, relative to the invoking interpreter. If the target interpreter for the alias is the invoking interpreter then an empty list is returned. If the target interpreter for the alias is not the invoking interpreter or one of its descendants then an error is generated. The target command does not have to be defined at the time of this invocation.

**interp transfer srcPath channelId destPath**
Causes the IO channel identified by channelId to become available
in the interpreter identified by `destPath` and unavailable in the interpreter identified by `srcPath`.

<table>
<thead>
<tr>
<th>SLAVE COMMAND</th>
</tr>
</thead>
</table>

For each slave interpreter created with the `interp` command, a new Tcl command is created in the master interpreter with the same name as the new interpreter. This command may be used to invoke various operations on the interpreter. It has the following general form:

```
slave command ?arg arg ...?
```

`Slave` is the name of the interpreter, and `command` and the `args` determine the exact behavior of the command. The valid forms of this command are:

**slave aliases**

- Returns a Tcl list whose elements are the tokens of all the aliases in `slave`. The tokens correspond to the values returned when the aliases were created (which may not be the same as the current names of the commands).

**slave alias srcToken**

- Returns a Tcl list whose elements are the `targetCmd` and `args` associated with the alias represented by `srcToken` (this is the value returned when the alias was created; it is possible that the actual source command in the slave is different from `srcToken`).

**slave alias srcToken {}**

- Deletes the alias for `srcToken` in the slave interpreter. `srcToken` refers to the value returned when the alias was created; if the source command has been renamed, the renamed command will be deleted.

**slave alias srcCmd targetCmd ?arg ..?**

- Creates an alias such that whenever `srcCmd` is invoked in `slave`, `targetCmd` is invoked in the master. The `arg` arguments will be
passed to `targetCmd` as additional arguments, prepended before any arguments passed in the invocation of `srcCmd`. See ALIAS INVOCATION below for details. The command returns a token that uniquely identifies the command created `srcCmd`, even if the command is renamed afterwards. The token may but does not have to be equal to `srcCmd`.

`slave bgerror ?cmdPrefix?`  
This command either gets or sets the current background error handler for the `slave` interpreter. If `cmdPrefix` is absent, the current background error handler is returned, and if it is present, it is a list of words (of minimum length one) that describes what to set the interpreter's background error to. See the BACKGROUND ERROR HANDLING section for more details.

`slave eval arg ?arg ..?`  
This command concatenates all of the `arg` arguments in the same fashion as the `concat` command, then evaluates the resulting string as a Tcl script in `slave`. The result of this evaluation (including all return options, such as `-errorinfo` and `-errorcode` information, if an error occurs) is returned to the invoking interpreter. Note that the script will be executed in the current context stack frame of `slave`; this is so that the implementations (in a master interpreter) of aliases in a slave interpreter can execute scripts in the slave that find out information about the slave's current state and stack frame.

`slave expose hiddenName ?exposedCmdName?`  
This command exposes the hidden command `hiddenName`, eventually bringing it back under a new `exposedCmdName` name (this name is currently accepted only if it is a valid global name space name without any ::), in `slave`. If an exposed command with the targeted name already exists, this command fails. For more details on hidden commands, see HIDDEN COMMANDS, below.

`slave hide exposedCmdName ?hiddenCmdName?`  
This command hides the exposed command `exposedCmdName`, renaming it to the hidden command `hiddenCmdName`, or keeping the same name if the argument is not given, in the `slave` interpreter.
If a hidden command with the targeted name already exists, this command fails. Currently both `exposedCmdName` and `hiddenCmdName` can not contain namespace qualifiers, or an error is raised. Commands to be hidden are looked up in the global namespace even if the current namespace is not the global one. This prevents slaves from fooling a master interpreter into hiding the wrong command, by making the current namespace be different from the global one. For more details on hidden commands, see **HIDDEN COMMANDS**, below.

`slave hidden`

Returns a list of the names of all hidden commands in `slave`.

`slave invokehidden ?-option ...? hiddenName ?arg ..?`

This command invokes the hidden command `hiddenName` with the supplied arguments, in `slave`. No substitutions or evaluations are applied to the arguments. Three `-options` are supported, all of which start with `-`: `namespace` (which takes a single argument afterwards, `nsName`), `global`, and `--`. If the `namespace` flag is given, the hidden command is invoked in the specified namespace in the slave. If the `global` flag is given, the command is invoked at the global level in the slave; otherwise it is invoked at the current call frame and can access local variables in that or outer call frames. The `--` flag allows the `hiddenCmdName` argument to start with a “-” character, and is otherwise unnecessary. If both the `namespace` and `global` flags are given, the `namespace` flag is ignored. Note that the hidden command will be executed (by default) in the current context stack frame of `slave`. For more details on hidden commands, see **HIDDEN COMMANDS**, below.

`slave issafe`

Returns `1` if the slave interpreter is safe, `0` otherwise.

`slave limit limitType ?-option? ?value ...?`

Sets up, manipulates and queries the configuration of the resource limit `limitType` for the slave interpreter. If no `-option` is specified, return the current configuration of the limit. If `-option` is the sole argument, return the value of that option. Otherwise, a list of -
option/value argument pairs must supplied. See RESOURCE LIMITS below for a more detailed explanation of what limits and options are supported.

**slave marktrusted**
Marks the slave interpreter as trusted. Can only be invoked by a trusted interpreter. This command does not expose any hidden commands in the slave interpreter. The command has no effect if the slave is already trusted.

**slave recursionlimit ?newlimit?**
Returns the maximum allowable nesting depth for the slave interpreter. If newlimit is specified, the recursion limit in slave will be set so that nesting of more than newlimit calls to Tcl_Eval() and related procedures in slave will return an error. The newlimit value is also returned. The newlimit value must be a positive integer between 1 and the maximum value of a non-long integer on the platform.

The command sets the maximum size of the Tcl call stack only. It cannot by itself prevent stack overflows on the C stack being used by the application. If your machine has a limit on the size of the C stack, you may get stack overflows before reaching the limit set by the command. If this happens, see if there is a mechanism in your system for increasing the maximum size of the C stack.

**SAFE INTERPRETERS**

A safe interpreter is one with restricted functionality, so that is safe to execute an arbitrary script from your worst enemy without fear of that script damaging the enclosing application or the rest of your computing environment. In order to make an interpreter safe, certain commands and variables are removed from the interpreter. For example, commands to create files on disk are removed, and the **exec** command is removed, since it could be used to cause damage through subprocesses. Limited access to these facilities can be provided, by creating aliases to the master interpreter which check their arguments carefully and provide restricted access to a safe subset of facilities. For
example, file creation might be allowed in a particular subdirectory and subprocess invocation might be allowed for a carefully selected and fixed set of programs.

A safe interpreter is created by specifying the `-safe` switch to the `interp create` command. Furthermore, any slave created by a safe interpreter will also be safe.

A safe interpreter is created with exactly the following set of built-in commands:

<table>
<thead>
<tr>
<th>after</th>
<th>append</th>
<th>apply</th>
<th>array</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>break</td>
<td>catch</td>
<td>chan</td>
</tr>
<tr>
<td>clock</td>
<td>close</td>
<td>concat</td>
<td>continue</td>
</tr>
<tr>
<td>dict</td>
<td>eof</td>
<td>error</td>
<td>eval</td>
</tr>
<tr>
<td>expr</td>
<td>fblocked</td>
<td>fcop y</td>
<td>fileevent</td>
</tr>
<tr>
<td>flush</td>
<td>for</td>
<td>foreach</td>
<td>format</td>
</tr>
<tr>
<td>gets</td>
<td>global</td>
<td>if</td>
<td>incr</td>
</tr>
<tr>
<td>info</td>
<td>interp</td>
<td>join</td>
<td>lappend</td>
</tr>
<tr>
<td>lassign</td>
<td>lindex</td>
<td>linset</td>
<td>list</td>
</tr>
<tr>
<td>Command</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>llength</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>llrange</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>llrepeat</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>llreplace</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lsearch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lsort</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>namespace</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>package</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>proc</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>puts</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>read</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>regexp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>regsub</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rename</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>return</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>seek</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>split</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>subst</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>switch</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tell</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trace</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>update</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uplevel</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>upvar</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vwait</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>while</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following commands are hidden by `interp create` when it creates a safe interpreter:

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
</tr>
<tr>
<td>encoding</td>
</tr>
<tr>
<td>exec</td>
</tr>
<tr>
<td>exit</td>
</tr>
<tr>
<td>fconfigure</td>
</tr>
<tr>
<td>file</td>
</tr>
<tr>
<td>glob</td>
</tr>
<tr>
<td>load</td>
</tr>
<tr>
<td>open</td>
</tr>
<tr>
<td>pwd</td>
</tr>
<tr>
<td>socket</td>
</tr>
<tr>
<td>source</td>
</tr>
</tbody>
</table>
These commands can be recreated later as Tcl procedures or aliases, or re-exposed by `interp expose`.

The following commands from Tcl's library of support procedures are not present in a safe interpreter:

- auto_exec_ok
- auto_import
- auto_load
- auto_load_index
- auto_qualify
- unknown
- auto_mkindex
- auto_mkindex_old
- auto_reset
- history
- parray
- pkg_mkIndex
- ::pkg::create
- ::safe::interpAddToAccessPath
- ::safe::interpCreate
- ::safe::interpConfigure
- ::safe::interpDelete
- ::safe::interpFindInAccessPath
can only be provided by explicit definition of an **unknown** command in
the safe interpreter. This will involve exposing the **source** command.
This is most easily accomplished by creating the safe interpreter with
Tcl's **Safe-Tcl** mechanism. **Safe-Tcl** provides safe versions of **source**, **load**, and other Tcl commands needed to support autoloading of
commands and the loading of packages.

In addition, the **env** variable is not present in a safe interpreter, so it
cannot share environment variables with other interpreters. The **env**
variable poses a security risk, because users can store sensitive
information in an environment variable. For example, the PGP manual
recommends storing the PGP private key protection password in the
environment variable **PGPPASS**. Making this variable available to
untrusted code executing in a safe interpreter would incur a security
risk.

If extensions are loaded into a safe interpreter, they may also restrict
their own functionality to eliminate unsafe commands. For a discussion
of management of extensions for safety see the manual entries for
**Safe-Tcl** and the **load** Tcl command.

A safe interpreter may not alter the recursion limit of any interpreter,
including itself.

**ALIAS INVOCATION**
The alias mechanism has been carefully designed so that it can be used safely when an untrusted script is executing in a safe slave and the target of the alias is a trusted master. The most important thing in guaranteeing safety is to ensure that information passed from the slave to the master is never evaluated or substituted in the master; if this were to occur, it would enable an evil script in the slave to invoke arbitrary functions in the master, which would compromise security.

When the source for an alias is invoked in the slave interpreter, the usual Tcl substitutions are performed when parsing that command. These substitutions are carried out in the source interpreter just as they would be for any other command invoked in that interpreter. The command procedure for the source command takes its arguments and merges them with the targetCmd and args for the alias to create a new array of arguments. If the words of srcCmd were "srcCmd arg1 arg2 ... argN", the new set of words will be “targetCmd arg arg ... arg arg1 arg2 ... arg arg1 arg2 ... argN”, where targetCmd and args are the values supplied when the alias was created. TargetCmd is then used to locate a command procedure in the target interpreter, and that command procedure is invoked with the new set of arguments. An error occurs if there is no command named targetCmd in the target interpreter. No additional substitutions are performed on the words: the target command procedure is invoked directly, without going through the normal Tcl evaluation mechanism. Substitutions are thus performed on each word exactly once: targetCmd and args were substituted when parsing the command that created the alias, and arg1 - argN are substituted when the alias's source command is parsed in the source interpreter.

When writing the targetCmds for aliases in safe interpreters, it is very important that the arguments to that command never be evaluated or substituted, since this would provide an escape mechanism whereby the slave interpreter could execute arbitrary code in the master. This in turn would compromise the security of the system.

| HIDDEN COMMANDS |

Safe interpreters greatly restrict the functionality available to Tcl programs executing within them. Allowing the untrusted Tcl program to
have direct access to this functionality is unsafe, because it can be used for a variety of attacks on the environment. However, there are times when there is a legitimate need to use the dangerous functionality in the context of the safe interpreter. For example, sometimes a program must be `source`ed into the interpreter. Another example is Tk, where windows are bound to the hierarchy of windows for a specific interpreter; some potentially dangerous functions, e.g. window management, must be performed on these windows within the interpreter context.

The `interp` command provides a solution to this problem in the form of hidden commands. Instead of removing the dangerous commands entirely from a safe interpreter, these commands are hidden so they become unavailable to Tcl scripts executing in the interpreter. However, such hidden commands can be invoked by any trusted ancestor of the safe interpreter, in the context of the safe interpreter, using `interp invoke`. Hidden commands and exposed commands reside in separate name spaces. It is possible to define a hidden command and an exposed command by the same name within one interpreter.

Hidden commands in a slave interpreter can be invoked in the body of procedures called in the master during alias invocation. For example, an alias for `source` could be created in a slave interpreter. When it is invoked in the slave interpreter, a procedure is called in the master interpreter to check that the operation is allowable (e.g. it asks to source a file that the slave interpreter is allowed to access). The procedure then it invokes the hidden `source` command in the slave interpreter to actually source in the contents of the file. Note that two commands named `source` exist in the slave interpreter: the alias, and the hidden command.

Because a master interpreter may invoke a hidden command as part of handling an alias invocation, great care must be taken to avoid evaluating any arguments passed in through the alias invocation. Otherwise, malicious slave interpreters could cause a trusted master interpreter to execute dangerous commands on their behalf. See the section on `ALIAS INVOCATION` for a more complete discussion of this topic. To help avoid this problem, no substitutions or evaluations are
applied to arguments of interp invokehidden.

Safe interpreters are not allowed to invoke hidden commands in themselves or in their descendants. This prevents safe slaves from gaining access to hidden functionality in themselves or their descendants.

The set of hidden commands in an interpreter can be manipulated by a trusted interpreter using interp expose and interp hide. The interp expose command moves a hidden command to the set of exposed commands in the interpreter identified by path, potentially renaming the command in the process. If an exposed command by the targeted name already exists, the operation fails. Similarly, interp hide moves an exposed command to the set of hidden commands in that interpreter. Safe interpreters are not allowed to move commands between the set of hidden and exposed commands, in either themselves or their descendants.

Currently, the names of hidden commands cannot contain namespace qualifiers, and you must first rename a command in a namespace to the global namespace before you can hide it. Commands to be hidden by interp hide are looked up in the global namespace even if the current namespace is not the global one. This prevents slaves from fooling a master interpreter into hiding the wrong command, by making the current namespace be different from the global one.

---

**RESOURCE LIMITS**

Every interpreter has two kinds of resource limits that may be imposed by any master interpreter upon its slaves. Command limits (of type command) restrict the total number of Tcl commands that may be executed by an interpreter (as can be inspected via the info cmdcount command), and time limits (of type time) place a limit by which execution within the interpreter must complete. Note that time limits are expressed as absolute times (as in clock seconds) and not relative times (as in after) because they may be modified after creation.

When a limit is exceeded for an interpreter, first any handler callbacks
defined by master interpreters are called. If those callbacks increase or remove the limit, execution within the (previously) limited interpreter continues. If the limit is still in force, an error is generated at that point and normal processing of errors within the interpreter (by the `catch` command) is disabled, so the error propagates outwards (building a stack-trace as it goes) to the point where the limited interpreter was invoked (e.g. by `interp eval`) where it becomes the responsibility of the calling code to catch and handle.

**LIMIT OPTIONS**

Every limit has a number of options associated with it, some of which are common across all kinds of limits, and others of which are particular to the kind of limit.

**-command**

This option (common for all limit types) specifies (if non-empty) a Tcl script to be executed in the global namespace of the interpreter reading and writing the option when the particular limit in the limited interpreter is exceeded. The callback may modify the limit on the interpreter if it wishes the limited interpreter to continue executing. If the callback generates an error, it is reported through the background error mechanism (see BACKGROUND ERROR HANDLING). Note that the callbacks defined by one interpreter are completely isolated from the callbacks defined by another, and that the order in which those callbacks are called is undefined.

**-granularity**

This option (common for all limit types) specifies how frequently (out of the points when the Tcl interpreter is in a consistent state where limit checking is possible) that the limit is actually checked. This allows the tuning of how frequently a limit is checked, and hence how often the limit-checking overhead (which may be substantial in the case of time limits) is incurred.

**-milliseconds**

This option specifies the number of milliseconds after the moment
defined in the **-seconds** option that the time limit will fire. It should only ever be specified in conjunction with the **-seconds** option (whether it was set previously or is being set this invocation.)

**-seconds**

This option specifies the number of seconds after the epoch (see **clock seconds**) that the time limit for the interpreter will be triggered. The limit will be triggered at the start of the second unless specified at a sub-second level using the **-milliseconds** option. This option may be the empty string, which indicates that a time limit is not set for the interpreter.

**-value**

This option specifies the number of commands that the interpreter may execute before triggering the command limit. This option may be the empty string, which indicates that a command limit is not set for the interpreter.

Where an interpreter with a resource limit set on it creates a slave interpreter, that slave interpreter will have resource limits imposed on it that are at least as restrictive as the limits on the creating master interpreter. If the master interpreter of the limited master wishes to relax these conditions, it should hide the **interp** command in the child and then use aliases and the **interp invokehidden** subcommand to provide such access as it chooses to the **interp** command to the limited master as necessary.

### BACKGROUND ERROR HANDLING

When an error happens in a situation where it cannot be reported directly up the stack (e.g. when processing events in an **update** or **vwait** call) the error is instead reported through the background error handling mechanism. Every interpreter has a background error handler registered; the default error handler arranges for the **bgerror** command in the interpreter's global namespace to be called, but other error handlers may be installed and process background errors in substantially different ways.
A background error handler consists of a non-empty list of words to which will be appended two further words at invocation time. The first word will be the error message string, and the second will a dictionary of return options (this is also the sort of information that can be obtained by trapping a normal error using `catch` of course.) The resulting list will then be executed in the interpreter's global namespace without further substitutions being performed.

**CREDITS**

The safe interpreter mechanism is based on the Safe-Tcl prototype implemented by Nathaniel Borenstein and Marshall Rose.

**EXAMPLES**

Creating and using an alias for a command in the current interpreter:

```
interp alias {} getIndex {} lsearch {alpha beta gamma delta} set idx [getIndex delta]
```

Executing an arbitrary command in a safe interpreter where every invocation of `lappend` is logged:

```
set i [interp create -safe]
interp hide $i lappend
interp alias $i lappend {} loggedLappend $i
proc loggedLappend {i args}
  puts "logged invocation of lappend $args"
  interp invokehidden $i lappend {*}$args
interp eval $i $someUntrustedScript
```

Setting a resource limit on an interpreter so that an infinite loop terminates.
set i [interp create]
interp limit $i command -value 1000
interp eval $i {
    set x 0
    while {1} {
        puts "Counting up... [incr x]"
    }
}

SEE ALSO
bgerror, load, safe, Tcl_CreateSlave

KEYWORDS
alias, master interpreter, safe interpreter, slave interpreter

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
Copyright © 2004 Donal K. Fellows
NAME
pkg::create - Construct an appropriate 'package ifneeded' command for a given package specification

SYNOPSIS
DESCRIPTION
OPTIONS
   -name packageName
   -version packageVersion
   -load filespec
   -source filespec

SEE ALSO
KEYWORDS

NAME
pkg::create - Construct an appropriate 'package ifneeded' command for a given package specification

SYNOPSIS
::pkg::create -name packageName -version packageVersion ?-load filespec? ... ?-source filespec? ...

DESCRIPTION
::pkg::create is a utility procedure that is part of the standard Tcl library. It is used to create an appropriate package ifneeded command for a given package specification. It can be used to construct a pkgIndex.tcl file for use with the package mechanism.

OPTIONS
The parameters supported are:

- **-name packageName**
  This parameter specifies the name of the package. It is required.

- **-version packageVersion**
  This parameter specifies the version of the package. It is required.

- **-load filespec**
  This parameter specifies a binary library that must be loaded with the **load** command. *filespec* is a list with two elements. The first element is the name of the file to load. The second, optional element is a list of commands supplied by loading that file. If the list of procedures is empty or omitted, **::pkg::create** will set up the library for direct loading (see **pkg_mkIndex**). Any number of **-load** parameters may be specified.

- **-source filespec**
  This parameter is similar to the **-load** parameter, except that it specifies a Tcl library that must be loaded with the **source** command. Any number of **-source** parameters may be specified.

At least one **-load** or **-source** parameter must be given.

**SEE ALSO**

**package**

**KEYWORDS**

**auto-load, index, package, version**

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
   Tcl - Tool Command Language

SYNOPSIS

DESCRIPTION
   $\text{name}$
   $\text{name}(\text{index})$
   ${\text{name}}$
   \a
   \b
   \f
   \n
   \r
   \t
   \v
   \<newline>\text{whiteSpace}
   \|
   \ooo
   \xhh
   \uhhh
[12] Substitution and word boundaries.
NAME

Tcl - Tool Command Language

SYNOPSIS

Summary of Tcl language syntax.

DESCRIPTION

The following rules define the syntax and semantics of the Tcl language:

A Tcl script is a string containing one or more commands. Semi-
colons and newlines are command separators unless quoted as
described below. Close brackets are command terminators during
command substitution (see below) unless quoted.

A command is evaluated in two steps. First, the Tcl interpreter
breaks the command into words and performs substitutions as
described below. These substitutions are performed in the same
way for all commands. The first word is used to locate a command
procedure to carry out the command, then all of the words of the
command are passed to the command procedure. The command
procedure is free to interpret each of its words in any way it likes,
such as an integer, variable name, list, or Tcl script. Different
commands interpret their words differently.

Words of a command are separated by white space (except for
newlines, which are command separators).

If the first character of a word is double-quote ("""") then the word is
terminated by the next double-quote character. If semi-colons,
close brackets, or white space characters (including newlines)
appear between the quotes then they are treated as ordinary
characters and included in the word. Command substitution, variable substitution, and backslash substitution are performed on the characters between the quotes as described below. The double-quotes are not retained as part of the word.

[5] **Argument expansion.**
If a word starts with the string “{*}” followed by a non-whitespace character, then the leading “{*}” is removed and the rest of the word is parsed and substituted as any other word. After substitution, the word is parsed again without substitutions, and its words are added to the command being substituted. For instance, “cmd a {*}{b c} d {*}{e f}” is equivalent to “cmd a b c d e f”.

[6] **Braces.**
If the first character of a word is an open brace (“{“) and rule [5] does not apply, then the word is terminated by the matching close brace (“}”). Braces nest within the word: for each additional open brace there must be an additional close brace (however, if an open brace or close brace within the word is quoted with a backslash then it is not counted in locating the matching close brace). No substitutions are performed on the characters between the braces except for backslash-newline substitutions described below, nor do semi-colons, newlines, close brackets, or white space receive any special interpretation. The word will consist of exactly the characters between the outer braces, not including the braces themselves.

[7] **Command substitution.**
If a word contains an open bracket (“[“) then Tcl performs *command substitution*. To do this it invokes the Tcl interpreter recursively to process the characters following the open bracket as a Tcl script. The script may contain any number of commands and must be terminated by a close bracket (“]”). The result of the script (i.e. the result of its last command) is substituted into the word in place of the brackets and all of the characters between them. There may be any number of command substitutions in a single word. Command substitution is not performed on words enclosed in braces.
Variable substitution.
If a word contains a dollar-sign (“$”) followed by one of the forms described below, then Tcl performs variable substitution: the dollar-sign and the following characters are replaced in the word by the value of a variable. Variable substitution may take any of the following forms:

$name
Name is the name of a scalar variable; the name is a sequence of one or more characters that are a letter, digit, underscore, or namespace separators (two or more colons).

$name(index)
Name gives the name of an array variable and index gives the name of an element within that array. Name must contain only letters, digits, underscores, and namespace separators, and may be an empty string. Command substitutions, variable substitutions, and backslash substitutions are performed on the characters of index.

${name}
Name is the name of a scalar variable. It may contain any characters whatsoever except for close braces.

There may be any number of variable substitutions in a single word. Variable substitution is not performed on words enclosed in braces.

Backslash substitution.
If a backslash (“\”) appears within a word then backslash substitution occurs. In all cases but those described below the backslash is dropped and the following character is treated as an ordinary character and included in the word. This allows characters such as double quotes, close brackets, and dollar signs to be included in words without triggering special processing. The following table lists the backslash sequences that are handled specially, along with the value that replaces each sequence.
Audible alert (bell) (0x7).

Backspace (0x8).

Form feed (0xc).

Newline (0xa).

Carriage-return (0xd).

Tab (0x9).

Vertical tab (0xb).

\<newline\> \textit{whiteSpace}

A single space character replaces the backslash, newline, and all spaces and tabs after the newline. This backslash sequence is unique in that it is replaced in a separate pre-pass before the command is actually parsed. This means that it will be replaced even when it occurs between braces, and the resulting space will be treated as a word separator if it is not in braces or quotes.

Backslash ("\`).

The digits \textit{ooo} (one, two, or three of them) give an eight-bit octal value for the Unicode character that will be inserted. The upper bits of the Unicode character will be 0.
The hexadecimal digits $hh$ give an eight-bit hexadecimal value for the Unicode character that will be inserted. Any number of hexadecimal digits may be present; however, all but the last two are ignored (the result is always a one-byte quantity). The upper bits of the Unicode character will be 0.

\uhhhh

The hexadecimal digits $hhhh$ (one, two, three, or four of them) give a sixteen-bit hexadecimal value for the Unicode character that will be inserted.

Backslash substitution is not performed on words enclosed in braces, except for backslash-newline as described above.

If a hash character (“#”) appears at a point where Tcl is expecting the first character of the first word of a command, then the hash character and the characters that follow it, up through the next newline, are treated as a comment and ignored. The comment character only has significance when it appears at the beginning of a command.

Each character is processed exactly once by the Tcl interpreter as part of creating the words of a command. For example, if variable substitution occurs then no further substitutions are performed on the value of the variable; the value is inserted into the word verbatim. If command substitution occurs then the nested command is processed entirely by the recursive call to the Tcl interpreter; no substitutions are performed before making the recursive call and no additional substitutions are performed on the result of the nested script.

Substitutions take place from left to right, and each substitution is evaluated completely before attempting to evaluate the next. Thus, a sequence like
set y [set x 0][incr x][incr x]

will always set the variable y to the value, 012.

[12] **Substitution and word boundaries.**
Substitutions do not affect the word boundaries of a command, except for argument expansion as specified in rule [5]. For example, during variable substitution the entire value of the variable becomes part of a single word, even if the variable's value contains spaces.
NAME
eof - Check for end of file condition on channel

SYNOPSIS
eof channelId

DESCRIPTION
Returns 1 if an end of file condition occurred during the most recent input operation on channelId (such as gets), 0 otherwise.

ChannelId must be an identifier for an open channel such as a Tcl standard channel (stdin, stdout, or stderr), the return value from an invocation of open or socket, or the result of a channel creation command provided by a Tcl extension.

EXAMPLES
Read and print out the contents of a file line-by-line:

```
set f [open somefile.txt]
while {1} {
    set line [gets $f]
    if {[eof $f]} {
        close $f
        break
    }
    puts "Read line: $line"
}
```
Read and print out the contents of a file by fixed-size records:

```tcl
set f [open somefile.dat]
fconfigure $f -translation binary
set recordSize 40
while {1} {
    set record [read $f $recordSize]
    if {[eof $f]} {
        close $f
        break
    } 
    puts "Read record: $record"
}
```

SEE ALSO

- file
- open
- close
- fblocked
- Tcl_StandardChannels

KEYWORDS

- channel
- end of file
NAME
join - Create a string by joining together list elements

SYNOPSIS
join list ?joinString?

DESCRIPTION
The list argument must be a valid Tcl list. This command returns the string formed by joining all of the elements of list together with joinString separating each adjacent pair of elements. The joinString argument defaults to a space character.

EXAMPLES
Making a comma-separated list:

    set data {1 2 3 4 5}
    join $data ", "
    → 1, 2, 3, 4, 5

Using join to flatten a list by a single level:

    set data {1 {2 3} 4 {5 {6 7} 8}}
    join $data
    → 1 2 3 4 5 {6 7} 8
SEE ALSO

list, lappend, split

KEYWORDS

element, join, list, separator

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
pkg_mkIndex - Build an index for automatic loading of packages

SYNOPSIS

DESCRIPTION

OPTIONS
-direct
-lazy
-load pkgPat
-verbose
--

PACKAGES AND THE AUTO-LOADER

HOW IT WORKS

DIRECT LOADING

COMPLEX CASES

SEE ALSO

KEYWORDS

NAME

pkg_mkIndex - Build an index for automatic loading of packages

SYNOPSIS


DESCRIPTION

Pkg_mkIndex is a utility procedure that is part of the standard Tcl library. It is used to create index files that allow packages to be loaded automatically when package require commands are executed. To use
pkg_mkIndex, follow these steps:

[1]
Create the package(s). Each package may consist of one or more Tcl script files or binary files. Binary files must be suitable for loading with the load command with a single argument; for example, if the file is test.so it must be possible to load this file with the command load test.so. Each script file must contain a package provide command to declare the package and version number, and each binary file must contain a call to Tcl_PkgProvide.

[2]
Create the index by invoking pkg_mkIndex. The dir argument gives the name of a directory and each pattern argument is a glob-style pattern that selects script or binary files in dir. The default pattern is *.tcl and *.info sharedlibextension].

Pkg_mkIndex will create a file pkgIndex.tcl in dir with package information about all the files given by the pattern arguments. It does this by loading each file into a slave interpreter and seeing what packages and new commands appear (this is why it is essential to have package provide commands or Tcl_PkgProvide calls in the files, as described above). If you have a package split among scripts and binary files, or if you have dependencies among files, you may have to use the -load option or adjust the order in which pkg_mkIndex processes the files. See COMPLEX CASES below.

[3]
Install the package as a subdirectory of one of the directories given by the tcl_pkgPath variable. If $tcl_pkgPath contains more than one directory, machine-dependent packages (e.g., those that contain binary shared libraries) should normally be installed under the first directory and machine-independent packages (e.g., those that contain only Tcl scripts) should be installed under the second directory. The subdirectory should include the package's script and/or binary files as well as the pkgIndex.tcl file. As long as the
package is installed as a subdirectory of a directory in $tcl_pkgPath it will automatically be found during package require commands.

If you install the package anywhere else, then you must ensure that the directory containing the package is in the auto_path global variable or an immediate subdirectory of one of the directories in auto_path. Auto_path contains a list of directories that are searched by both the auto-loader and the package loader; by default it includes $tcl_pkgPath. The package loader also checks all of the subdirectories of the directories in auto_path. You can add a directory to auto_path explicitly in your application, or you can add the directory to your TCLLIBPATH environment variable: if this environment variable is present, Tcl initializes auto_path from it during application startup.

[4] Once the above steps have been taken, all you need to do to use a package is to invoke package require. For example, if versions 2.1, 2.3, and 3.1 of package Test have been indexed by pkg_mkIndex, the command package require Test will make version 3.1 available and the command package require -exact Test 2.1 will make version 2.1 available. There may be many versions of a package in the various index files in auto_path, but only one will actually be loaded in a given interpreter, based on the first call to package require. Different versions of a package may be loaded in different interpreters.

OPTIONS

The optional switches are:

- **direct**
  
The generated index will implement direct loading of the package upon package require. This is the default.

- **lazy**
  
The generated index will manage to delay loading the package until
the use of one of the commands provided by the package, instead of loading it immediately upon `package require`. This is not compatible with the use of `auto_reset`, and therefore its use is discouraged.

- **load `pkgPat`**
  The index process will pre-load any packages that exist in the current interpreter and match `pkgPat` into the slave interpreter used to generate the index. The pattern match uses string match rules, but without making case distinctions. See COMPLEX CASES below.

- **verbose**
  Generate output during the indexing process. Output is via the `tclLog` procedure, which by default prints to stderr.

--

End of the flags, in case `dir` begins with a dash.

---

**PACKAGES AND THE AUTO-LOADER**

The package management facilities overlap somewhat with the auto-loader, in that both arrange for files to be loaded on-demand. However, package management is a higher-level mechanism that uses the auto-loader for the last step in the loading process. It is generally better to index a package with `pkg_mkIndex` rather than `auto_mkindex` because the package mechanism provides version control: several versions of a package can be made available in the index files, with different applications using different versions based on `package require` commands. In contrast, `auto_mkindex` does not understand versions so it can only handle a single version of each package. It is probably not a good idea to index a given package with both `pkg_mkIndex` and `auto_mkindex`. If you use `pkg_mkIndex` to index a package, its commands cannot be invoked until `package require` has been used to select a version; in contrast, packages indexed with `auto_mkindex` can be used immediately since there is no version control.
HOW IT WORKS

Pkg_mkIndex depends on the **package unknown** command, the **package ifneeded** command, and the auto-loader. The first time a **package require** command is invoked, the **package unknown** script is invoked. This is set by Tcl initialization to a script that evaluates all of the pkgIndex.tcl files in the **auto_path**. The pkgIndex.tcl files contain **package ifneeded** commands for each version of each available package; these commands invoke **package provide** commands to announce the availability of the package, and they setup auto-loader information to load the files of the package. If the `-lazy` flag was provided when the pkgIndex.tcl was generated, a given file of a given version of a given package is not actually loaded until the first time one of its commands is invoked. Thus, after invoking **package require** you may not see the package's commands in the interpreter, but you will be able to invoke the commands and they will be auto-loaded.

DIRECT LOADING

Some packages, for instance packages which use namespaces and export commands or those which require special initialization, might select that their package files be loaded immediately upon **package require** instead of delaying the actual loading to the first use of one of the package's command. This is the default mode when generating the package index. It can be overridden by specifying the `-lazy` argument.

COMPLEX CASES

Most complex cases of dependencies among scripts and binary files, and packages being split among scripts and binary files are handled OK. However, you may have to adjust the order in which files are processed by **pkg_mkIndex**. These issues are described in detail below.

If each script or file contains one package, and packages are only contained in one file, then things are easy. You simply specify all files to be indexed in any order with some glob patterns.
In general, it is OK for scripts to have dependencies on other packages. If scripts contain `package require` commands, these are stubbed out in the interpreter used to process the scripts, so these do not cause problems. If scripts call into other packages in global code, these calls are handled by a stub `unknown` command. However, if scripts make variable references to other package's variables in global code, these will cause errors. That is also bad coding style.

If binary files have dependencies on other packages, things can become tricky because it is not possible to stub out C-level APIs such as `Tcl_PkgRequire` API when loading a binary file. For example, suppose the BLT package requires Tk, and expresses this with a call to `Tcl_PkgRequire` in its `Blt_Init` routine. To support this, you must run `pkg_mkIndex` in an interpreter that has Tk loaded. You can achieve this with the `-load pkgPat` option. If you specify this option, `pkg_mkIndex` will load any packages listed by `info loaded` and that match `pkgPat` into the interpreter used to process files. In most cases this will satisfy the `Tcl_PkgRequire` calls made by binary files.

If you are indexing two binary files and one depends on the other, you should specify the one that has dependencies last. This way the one without dependencies will get loaded and indexed, and then the package it provides will be available when the second file is processed. You may also need to load the first package into the temporary interpreter used to create the index by using the `-load` flag; it will not hurt to specify package patterns that are not yet loaded.

If you have a package that is split across scripts and a binary file, then you should avoid the `-load` flag. The problem is that if you load a package before computing the index it masks any other files that provide part of the same package. If you must use `-load`, then you must specify the scripts first; otherwise the package loaded from the binary file may mask the package defined by the scripts.

**SEE ALSO**

`package`
NAME

bell - Ring a display's bell

SYNOPSIS

bell ?-displayof window? ?-nice?

DESCRIPTION

This command rings the bell on the display for window and returns an empty string. If the -displayof option is omitted, the display of the application's main window is used by default. The command uses the current bell-related settings for the display, which may be modified with programs such as xset.

If -nice is not specified, this command also resets the screen saver for the screen. Some screen savers will ignore this, but others will reset so that the screen becomes visible again.

KEYWORDS

beep, bell, ring
**NAME**
font - Create and inspect fonts.

**SYNOPSIS**

font actual font ?-displayof window? ?option? ?--? ?char?
font configure fontname ?option? ?value option value ...?
font create ?fontname? ?option value ...?
font delete fontname ?fontname ...?
font families ?-displayof window?
font measure font ?-displayof window? text
font metrics font ?-displayof window? ?option?
font names

**FONT DESCRIPTION**

[1] fontname
[2] systemfont
[4] X-font names (XLFD)
[5] option value ?option value ...?

**FONT METRICS**

-ascent
-descent
-linespace
-fixed

**FONT OPTIONS**

-family name
-size size
-weight weight
-slant slant
-underline boolean
-overstrike boolean

**STANDARD FONTS**
font

font option ?arg arg ...?  

The font command provides several facilities for dealing with fonts, such as defining named fonts and inspecting the actual attributes of a font. The command has several different forms, determined by the first argument. The following forms are currently supported:

**font actual**  
font ?-displayof window? ?option? ?--? ?char?  

Returns information about the actual attributes that are obtained when font is used on window's display; the actual attributes obtained may differ from the attributes requested due to platform-dependent limitations, such as the availability of font families and pointsizes. font is a font description; see FONT DESCRIPTIONS.
below. If the `window` argument is omitted, it defaults to the main window. If `option` is specified, returns the value of that attribute; if it is omitted, the return value is a list of all the attributes and their values. See `FONT OPTIONS` below for a list of the possible attributes. If the `char` argument is supplied, it must be a single character. The font attributes returned will be those of the specific font used to render that character, which will be different from the base font if the base font does not contain the given character. If `char` may be a hyphen, it should be preceded by `--` to distinguish it from a misspelled `option`.

**font configure** `fontname` `option` `value` ...
Query or modify the desired attributes for the named font called `fontname`. If no `option` is specified, returns a list describing all the options and their values for `fontname`. If a single `option` is specified with no `value`, then returns the current value of that attribute. If one or more `option-value` pairs are specified, then the command modifies the given named font to have the given values; in this case, all widgets using that font will redisplay themselves using the new attributes for the font. See `FONT OPTIONS` below for a list of the possible attributes.

**font create** `fontname` `option` `value` ...
Creates a new named font and returns its name. `fontname` specifies the name for the font; if it is omitted, then Tk generates a new name of the form `fontx`, where `x` is an integer. There may be any number of `option-value` pairs, which provide the desired attributes for the new named font. See `FONT OPTIONS` below for a list of the possible attributes.

**font delete** `fontname` ...
Delete the specified named fonts. If there are widgets using the named font, the named font will not actually be deleted until all the instances are released. Those widgets will continue to display using the last known values for the named font. If a deleted named font is subsequently recreated with another call to `font create`, the widgets will use the new named font and redisplay themselves using the new attributes of that font.
**font families** `-displayof window`?
The return value is a list of the case-insensitive names of all font families that exist on window's display. If the window argument is omitted, it defaults to the main window.

**font measure** `font` `-displayof window`? `text`
Measures the amount of space the string text would use in the given font when displayed in window. font is a font description; see **FONT DESCRIPTIONS** below. If the window argument is omitted, it defaults to the main window. The return value is the total width in pixels of text, not including the extra pixels used by highly exaggerated characters such as cursive “f”. If the string contains newlines or tabs, those characters are not expanded or treated specially when measuring the string.

**font metrics** `font` `-displayof window`? `?option`?
Returns information about the metrics (the font-specific data), for font when it is used on window's display. font is a font description; see **FONT DESCRIPTIONS** below. If the window argument is omitted, it defaults to the main window. If option is specified, returns the value of that metric; if it is omitted, the return value is a list of all the metrics and their values. See **FONT METRICS** below for a list of the possible metrics.

**font names**
The return value is a list of all the named fonts that are currently defined.

**FONT DESCRIPTION**

The following formats are accepted as a font description anywhere font is specified as an argument above; these same forms are also permitted when specifying the `-font` option for widgets.

[1] `fontname`
The name of a named font, created using the **font create** command. When a widget uses a named font, it is guaranteed that
this will never cause an error, as long as the named font exists, no matter what potentially invalid or meaningless set of attributes the named font has. If the named font cannot be displayed with exactly the specified attributes, some other close font will be substituted automatically.

[2] systemfont
The platform-specific name of a font, interpreted by the graphics server. This also includes, under X, an XLFD (see [4]) for which a single “*” character was used to elide more than one field in the middle of the name. See PLATFORM-SPECIFIC issues for a list of the system fonts.

A properly formed list whose first element is the desired font family and whose optional second element is the desired size. The interpretation of the size attribute follows the same rules described for -size in FONT OPTIONS below. Any additional optional arguments following the size are font styles. Possible values for the style arguments are as follows:

    normal      bold      roman   italic

    underline   overstrike

[4] X-font names (XLFD)
A Unix-centric font name of the form -foundry-family-weight-slant-setwidth-addstyle-pixel-point-resx-resy-spacing-width-charset-encoding. The “*” character may be used to skip individual fields that the user does not care about. There must be exactly one “*” for each field skipped, except that a “*” at the end of the XLFD skips any remaining fields; the shortest valid XLFD is simply “*”, signifying all fields as defaults. Any fields that were skipped are given default values. For compatibility, an XLFD always chooses a font of the specified pixel size (not point size); although this interpretation is not strictly correct, all existing applications using
XLFDs assumed that one “point” was in fact one pixel and would display incorrectly (generally larger) if the correct size font were actually used.

[5] *option value* ?*option value* ...?
A properly formed list of *option-value* pairs that specify the desired attributes of the font, in the same format used when defining a named font; see **FONT OPTIONS** below.

When font description *font* is used, the system attempts to parse the description according to each of the above five rules, in the order specified. Cases [1] and [2] must match the name of an existing named font or of a system font. Cases [3], [4], and [5] are accepted on all platforms and the closest available font will be used. In some situations it may not be possible to find any close font (e.g., the font family was a garbage value); in that case, some system-dependent default font is chosen. If the font description does not match any of the above patterns, an error is generated.

**FONT METRICS**

The following options are used by the *font metrics* command to query font-specific data determined when the font was created. These properties are for the whole font itself and not for individual characters drawn in that font. In the following definitions, the “baseline” of a font is the horizontal line where the bottom of most letters line up; certain letters, such as lower-case “g” stick below the baseline.

**-ascent**
The amount in pixels that the tallest letter sticks up above the baseline of the font, plus any extra blank space added by the designer of the font.

**-descent**
The largest amount in pixels that any letter sticks down below the baseline of the font, plus any extra blank space added by the designer of the font.
-linespace

Returns how far apart vertically in pixels two lines of text using the same font should be placed so that none of the characters in one line overlap any of the characters in the other line. This is generally the sum of the ascent above the baseline line plus the descent below the baseline.

-fixed

Returns a boolean flag that is “1” if this is a fixed-width font, where each normal character is the same width as all the other characters, or is “0” if this is a proportionally-spaced font, where individual characters have different widths. The widths of control characters, tab characters, and other non-printing characters are not included when calculating this value.

FONT OPTIONS

The following options are supported on all platforms, and are used when constructing a named font or when specifying a font using style [5] as above:

-family name

The case-insensitive font family name. Tk guarantees to support the font families named Courier (a monospaced “typewriter” font), Times (a serifed “newspaper” font), and Helvetica (a sans-serif “European” font). The most closely matching native font family will automatically be substituted when one of the above font families is used. The name may also be the name of a native, platform-specific font family; in that case it will work as desired on one platform but may not display correctly on other platforms. If the family is unspecified or unrecognized, a platform-specific default font will be chosen.

-size size

The desired size of the font. If the size argument is a positive number, it is interpreted as a size in points. If size is a negative number, its absolute value is interpreted as a size in pixels. If a font cannot be displayed at the specified size, a nearby size will be
chosen. If size is unspecified or zero, a platform-dependent default size will be chosen.

Sizes should normally be specified in points so the application will remain the same ruler size on the screen, even when changing screen resolutions or moving scripts across platforms. However, specifying pixels is useful in certain circumstances such as when a piece of text must line up with respect to a fixed-size bitmap. The mapping between points and pixels is set when the application starts, based on properties of the installed monitor, but it can be overridden by calling the \texttt{tk scaling} command.

-\texttt{-weight weight}
  The nominal thickness of the characters in the font. The value \texttt{normal} specifies a normal weight font, while \texttt{bold} specifies a bold font. The closest available weight to the one specified will be chosen. The default weight is \texttt{normal}.

-\texttt{-slant slant}
  The amount the characters in the font are slanted away from the vertical. Valid values for slant are \texttt{roman} and \texttt{italic}. A roman font is the normal, upright appearance of a font, while an italic font is one that is tilted some number of degrees from upright. The closest available slant to the one specified will be chosen. The default slant is \texttt{roman}.

-\texttt{-underline boolean}
  The value is a boolean flag that specifies whether characters in this font should be underlined. The default value for underline is \texttt{false}.

-\texttt{-overstrike boolean}
  The value is a boolean flag that specifies whether a horizontal line should be drawn through the middle of characters in this font. The default value for overstrike is \texttt{false}.

\textbf{STANDARD FONTS}

The following named fonts are supported on all systems, and default to
values that match appropriate system defaults.

TkDefaultFont
This font is the default for all GUI items not otherwise specified.

TkTextFont
This font should be used for user text in entry widgets, listboxes etc.

TkFixedFont
This font is the standard fixed-width font.

TkMenuFont
This font is used for menu items.

TkHeadingFont
This font should be used for column headings in lists and tables.

TkCaptionFont
This font should be used for window and dialog caption bars.

TkSmallCaptionFont
This font should be used for captions on contained windows or tool dialogs.

TkIconFont
This font should be used for icon captions.

TkTooltipFont
This font should be used for tooltip windows (transient information windows).

It is not advised to change these fonts, as they may be modified by Tk itself in response to system changes. Instead, make a copy of the font and modify that.

PLATFORM-SPECIFIC FONTS

The following system fonts are supported:
X Windows
All valid X font names, including those listed by xlsfonts(1), are available.

MS Windows
The following fonts are supported, and are mapped to the user's style defaults.

  system       ansi       device

  systemfixed  ansifixed  oemfixed

Mac OS X
The following fonts are supported, and are mapped to the user's style defaults.

  system       application  menu

Additionally, the following named fonts provide access to the Aqua theme fonts:

  systemSystemFont       systemEmphasize

  systemSmallSystemFont  systemSmallEmpl

  systemApplicationFont  systemLabelFont

  systemViewsFont        systemMenuTitleF

  systemMenuItemFont     systemMenuItemM
EXAMPLE

Fill a text widget with lots of font demonstrators, one for every font family installed on your system:

```tcl
pack [text .t -wrap none] -fill both -expand 1
set count 0
set tabwidth 0
foreach family [lsort -dictionary [font families]] {
    .t tag configure f[incr count] -font [list $family 10]
    .t insert end ${family}: {}
    "This is a simple sampler\n"
    .t configure -tags $family:
    set w [font measure [.t cget -font] ${family}:
    if {$w+5 > $tabwidth} {
        set tabwidth [expr {$w+5}]
        .t configure -tabs $tabwidth
    }
}
```

SEE ALSO
options

KEYWORDS

font

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
Copyright © 2006-2007 Daniel A. Steffen <das(at)users.sourceforge.net>
NAME

options - Standard options supported by widgets

DESCRIPTION

-activebackground, activeBackground, Foreground
-activeborderwidth, activeBorderWidth, BorderWidth
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-compound, compound, Compound
-disabledforeground, disabledForeground, DisabledForeground
-exportselection, exportSelection, ExportSelection
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-insertbackground, insertBackground, Foreground
-insertborderwidth, insertBorderWidth, BorderWidth
-insertofftime, insertOffTime, OffTime
-insertontime, insertOnTime, OnTime
-insertwidth, insertWidth, InsertWidth
-jump, jump, Jump
-justify, justify, Justify
-orien, orient, Orient
-padx, padX, Pad
-pady, padY, Pad
SEE ALSO
KEYWORDS

NAME

options - Standard options supported by widgets

DESCRIPTION

This manual entry describes the common configuration options supported by widgets in the Tk toolkit. Every widget does not necessarily support every option (see the manual entries for individual widgets for a list of the standard options supported by that widget), but if a widget does support an option with one of the names listed below, then the option has exactly the effect described below.

In the descriptions below, “Command-Line Name” refers to the switch used in class commands and **configure** widget commands to set this value. For example, if an option's command-line switch is **-foreground** and there exists a widget `.a.b.c`, then the command

```
.a.b.c configure -foreground black
```
may be used to specify the value **black** for the option in the widget .a.b.c. Command-line switches may be abbreviated, as long as the abbreviation is unambiguous. “Database Name” refers to the option's name in the option database (e.g. in .Xdefaults files). “Database Class” refers to the option's class value in the option database.

**Command-Line Name:** `-activebackground`
**Database Name:** `activeBackground`
**Database Class:** *Foreground*

Specifies background color to use when drawing active elements. An element (a widget or portion of a widget) is active if the mouse cursor is positioned over the element and pressing a mouse button will cause some action to occur. If strict Motif compliance has been requested by setting the `tk_strictMotif` variable, this option will normally be ignored; the normal background color will be used instead. For some elements on Windows and Macintosh systems, the active color will only be used while mouse button 1 is pressed over the element.

**Command-Line Name:** `-activeborderwidth`
**Database Name:** `activeBorderWidth`
**Database Class:** *BorderWidth*

Specifies a non-negative value indicating the width of the 3-D border drawn around active elements. See above for definition of active elements. The value may have any of the forms acceptable to `Tk_GetPixels`. This option is typically only available in widgets displaying more than one element at a time (e.g. menus but not buttons).

**Command-Line Name:** `-activeforeground`
**Database Name:** `activeForeground`
**Database Class:** *Background*

Specifies foreground color to use when drawing active elements. See above for definition of active elements.

**Command-Line Name:** `-anchor`
Database Name: anchor
Database Class: Anchor
  Specifies how the information in a widget (e.g. text or a bitmap) is to be displayed in the widget. Must be one of the values n, ne, e, se, s, sw, w, nw, or center. For example, nw means display the information such that its top-left corner is at the top-left corner of the widget.

Command-Line Name: -background or -bg
Database Name: background
Database Class: Background
  Specifies the normal background color to use when displaying the widget.

Command-Line Name: -bitmap
Database Name: bitmap
Database Class: Bitmap
  Specifies a bitmap to display in the widget, in any of the forms acceptable to Tk_GetBitmap. The exact way in which the bitmap is displayed may be affected by other options such as anchor or justify. Typically, if this option is specified then it overrides other options that specify a textual value to display in the widget but this is controlled by the compound option; the bitmap option may be reset to an empty string to re-enable a text display. In widgets that support both bitmap and image options, image will usually override bitmap.

Command-Line Name: -borderwidth or -bd
Database Name: borderWidth
Database Class: BorderWidth
  Specifies a non-negative value indicating the width of the 3-D border to draw around the outside of the widget (if such a border is being drawn; the relief option typically determines this). The value may also be used when drawing 3-D effects in the interior of the widget. The value may have any of the forms acceptable to Tk_GetPixels.

Command-Line Name: -cursor
Database Name: **cursor**
Database Class: **Cursor**
  Specifies the mouse cursor to be used for the widget. The value may have any of the forms acceptable to `Tk_GetCursor`. In addition, if an empty string is specified, it indicates that the widget should defer to its parent for cursor specification.

Command-Line Name: **-compound**
Database Name: **compound**
Database Class: **Compound**
  Specifies if the widget should display text and bitmaps/images at the same time, and if so, where the bitmap/image should be placed relative to the text. Must be one of the values **none**, **bottom**, **top**, **left**, **right**, or **center**. For example, the (default) value **none** specifies that the bitmap or image should (if defined) be displayed instead of the text, the value **left** specifies that the bitmap or image should be displayed to the left of the text, and the value **center** specifies that the bitmap or image should be displayed on top of the text.

Command-Line Name: **-disabledforeground**
Database Name: **disabledForeground**
Database Class: **DisabledForeground**
  Specifies foreground color to use when drawing a disabled element. If the option is specified as an empty string (which is typically the case on monochrome displays), disabled elements are drawn with the normal foreground color but they are dimmed by drawing them with a stippled fill pattern.

Command-Line Name: **-exportselection**
Database Name: **exportSelection**
Database Class: **ExportSelection**
  Specifies whether or not a selection in the widget should also be the X selection. The value may have any of the forms accepted by `Tcl_GetBoolean`, such as **true**, **false**, **0**, **1**, **yes**, or **no**. If the selection is exported, then selecting in the widget deselects the current X selection, selecting outside the widget deselects any widget selection, and the widget will respond to selection retrieval
requests when it has a selection. The default is usually for widgets to export selections.

Command-Line Name: **-font**
Database Name: **font**
Database Class: **Font**
   Specifies the font to use when drawing text inside the widget. The value may have any of the forms described in the **font** manual page under **FONT DESCRIPTION**.

Command-Line Name: **-foreground or -fg**
Database Name: **foreground**
Database Class: **Foreground**
   Specifies the normal foreground color to use when displaying the widget.

Command-Line Name: **-highlightbackground**
Database Name: **highlightBackground**
Database Class: **HighlightBackground**
   Specifies the color to display in the traversal highlight region when the widget does not have the input focus.

Command-Line Name: **-highlightcolor**
Database Name: **highlightColor**
Database Class: **HighlightColor**
   Specifies the color to use for the traversal highlight rectangle that is drawn around the widget when it has the input focus.

Command-Line Name: **-highlightthickness**
Database Name: **highlightThickness**
Database Class: **HighlightThickness**
   Specifies a non-negative value indicating the width of the highlight rectangle to draw around the outside of the widget when it has the input focus. The value may have any of the forms acceptable to **Tk_GetPixels**. If the value is zero, no focus highlight is drawn around the widget.

Command-Line Name: **-image**
Database Name: **image**
Database Class: **Image**
   Specifies an image to display in the widget, which must have been created with the `image create` command. Typically, if the image option is specified then it overrides other options that specify a bitmap or textual value to display in the widget, though this is controlled by the `compound` option; the image option may be reset to an empty string to re-enable a bitmap or text display.

Command-Line Name: **-insertbackground**
Database Name: **insertBackground**
Database Class: **Foreground**
   Specifies the color to use as background in the area covered by the insertion cursor. This color will normally override either the normal background for the widget (or the selection background if the insertion cursor happens to fall in the selection).

Command-Line Name: **-insertborderwidth**
Database Name: **insertBorderWidth**
Database Class: **BorderWidth**
   Specifies a non-negative value indicating the width of the 3-D border to draw around the insertion cursor. The value may have any of the forms acceptable to [Tk GetPixels](https://docs.python.org/3/library/tk.html#Tk.GetPixels).

Command-Line Name: **-insertofftime**
Database Name: **insertOffTime**
Database Class: **OffTime**
   Specifies a non-negative integer value indicating the number of milliseconds the insertion cursor should remain “off” in each blink cycle. If this option is zero then the cursor does not blink: it is on all the time.

Command-Line Name: **-insertontime**
Database Name: **insertOnTime**
Database Class: **OnTime**
   Specifies a non-negative integer value indicating the number of milliseconds the insertion cursor should remain “on” in each blink cycle.
Command-Line Name: -insertwidth
Database Name: insertWidth
Database Class: InsertWidth
Specifies a value indicating the total width of the insertion cursor. The value may have any of the forms acceptable to Tk::GetPixels. If a border has been specified for the insertion cursor (using the insertBorderWidth option), the border will be drawn inside the width specified by the insertWidth option.

Command-Line Name: -jump
Database Name: jump
Database Class: Jump
For widgets with a slider that can be dragged to adjust a value, such as scrollbars, this option determines when notifications are made about changes in the value. The option's value must be a boolean of the form accepted by Tcl::GetBoolean. If the value is false, updates are made continuously as the slider is dragged. If the value is true, updates are delayed until the mouse button is released to end the drag; at that point a single notification is made (the value “jumps” rather than changing smoothly).

Command-Line Name: -justify
Database Name: justify
Database Class: Justify
When there are multiple lines of text displayed in a widget, this option determines how the lines line up with each other. Must be one of left, center, or right. Left means that the lines' left edges all line up, center means that the lines' centers are aligned, and right means that the lines' right edges line up.

Command-Line Name: -orient
Database Name: orient
Database Class: Orient
For widgets that can lay themselves out with either a horizontal or vertical orientation, such as scrollbars, this option specifies which orientation should be used. Must be either horizontal or vertical or an abbreviation of one of these.
Command-Line Name: **-padx**  
Database Name: **padX**  
Database Class: **Pad**  
Specifies a non-negative value indicating how much extra space to request for the widget in the X-direction. The value may have any of the forms acceptable to **Tk_GetPixels**. When computing how large a window it needs, the widget will add this amount to the width it would normally need (as determined by the width of the things displayed in the widget); if the geometry manager can satisfy this request, the widget will end up with extra internal space to the left and/or right of what it displays inside. Most widgets only use this option for padding text: if they are displaying a bitmap or image, then they usually ignore padding options.

Command-Line Name: **-pady**  
Database Name: **padY**  
Database Class: **Pad**  
Specifies a non-negative value indicating how much extra space to request for the widget in the Y-direction. The value may have any of the forms acceptable to **Tk_GetPixels**. When computing how large a window it needs, the widget will add this amount to the height it would normally need (as determined by the height of the things displayed in the widget); if the geometry manager can satisfy this request, the widget will end up with extra internal space above and/or below what it displays inside. Most widgets only use this option for padding text: if they are displaying a bitmap or image, then they usually ignore padding options.

Command-Line Name: **-relief**  
Database Name: **relief**  
Database Class: **Relief**  
Specifies the 3-D effect desired for the widget. Acceptable values are **raised**, **sunken**, **flat**, **ridge**, **solid**, and **groove**. The value indicates how the interior of the widget should appear relative to its exterior; for example, **raised** means the interior of the widget should appear to protrude from the screen, relative to the exterior of the widget.
Command-Line Name: **-repeatdelay**
Database Name: *repeatDelay*
Database Class: **RepeatDelay**
   Specifies the number of milliseconds a button or key must be held down before it begins to auto-repeat. Used, for example, on the up-and down-arrows in scrollbars.

Command-Line Name: **-repeatinterval**
Database Name: *repeatInterval*
Database Class: **RepeatInterval**
   Used in conjunction with *repeatDelay*: once auto-repeat begins, this option determines the number of milliseconds between auto-repeats.

Command-Line Name: **-selectbackground**
Database Name: *selectBackground*
Database Class: **Foreground**
   Specifies the background color to use when displaying selected items.

Command-Line Name: **-selectborderwidth**
Database Name: *selectBorderWidth*
Database Class: **BorderWidth**
   Specifies a non-negative value indicating the width of the 3-D border to draw around selected items. The value may have any of the forms acceptable to *Tk_GetPixels*.

Command-Line Name: **-selectforeground**
Database Name: *selectForeground*
Database Class: **Background**
   Specifies the foreground color to use when displaying selected items.

Command-Line Name: **-setgrid**
Database Name: *setGrid*
Database Class: **SetGrid**
   Specifies a boolean value that determines whether this widget controls the resizing grid for its top-level window. This option is
typically used in text widgets, where the information in the widget has a natural size (the size of a character) and it makes sense for the window's dimensions to be integral numbers of these units. These natural window sizes form a grid. If the setGrid option is set to true then the widget will communicate with the window manager so that when the user interactively resizes the top-level window that contains the widget, the dimensions of the window will be displayed to the user in grid units and the window size will be constrained to integral numbers of grid units. See the section GRIDDLED GEOMETRY MANAGEMENT in the \texttt{wm} manual entry for more details.

Command-Line Name: \texttt{-takefocus}
Database Name: \texttt{takeFocus}
Database Class: \texttt{TakeFocus}
Determine whether the window accepts the focus during keyboard traversal (e.g., Tab and Shift-Tab). Before setting the focus to a window, the traversal scripts consult the value of the \texttt{takeFocus} option. A value of \texttt{0} means that the window should be skipped entirely during keyboard traversal. \texttt{1} means that the window should receive the input focus as long as it is viewable (it and all of its ancestors are mapped). An empty value for the option means that the traversal scripts make the decision about whether or not to focus on the window: the current algorithm is to skip the window if it is disabled, if it has no key bindings, or if it is not viewable. If the value has any other form, then the traversal scripts take the value, append the name of the window to it (with a separator space), and evaluate the resulting string as a Tcl script. The script must return \texttt{0}, \texttt{1}, or an empty string: a \texttt{0} or \texttt{1} value specifies whether the window will receive the input focus, and an empty string results in the default decision described above. Note: this interpretation of the option is defined entirely by the Tcl scripts that implement traversal: the widget implementations ignore the option entirely, so you can change its meaning if you redefine the keyboard traversal scripts.

Command-Line Name: \texttt{-text}
Database Name: **text**
Database Class: **Text**
   Specifies a string to be displayed inside the widget. The way in which the string is displayed depends on the particular widget and may be determined by other options, such as anchor or justify.

Command-Line Name: `-textvariable`
Database Name: **textVariable**
Database Class: **Variable**
   Specifies the name of a variable. The value of the variable is a text string to be displayed inside the widget; if the variable value changes then the widget will automatically update itself to reflect the new value. The way in which the string is displayed in the widget depends on the particular widget and may be determined by other options, such as anchor or justify.

Command-Line Name: `-troughcolor`
Database Name: **troughColor**
Database Class: **Background**
   Specifies the color to use for the rectangular trough areas in widgets such as scrollbars and scales. This option is ignored for scrollbars on Windows (native widget does not recognize this option).

Command-Line Name: `-underline`
Database Name: **underline**
Database Class: **Underline**
   Specifies the integer index of a character to underline in the widget. This option is used by the default bindings to implement keyboard traversal for menu buttons and menu entries. 0 corresponds to the first character of the text displayed in the widget, 1 to the next character, and so on.

Command-Line Name: `-wraplength`
Database Name: **wrapLength**
Database Class: **WrapLength**
   For widgets that can perform word-wrapping, this option specifies the maximum line length. Lines that would exceed this length are
wrapped onto the next line, so that no line is longer than the specified length. The value may be specified in any of the standard forms for screen distances. If this value is less than or equal to 0 then no wrapping is done: lines will break only at newline characters in the text.

Command-Line Name: -xscrollcommand
Database Name: xScrollCommand
Database Class: ScrollCommand
Specifies the prefix for a command used to communicate with horizontal scrollbars. When the view in the widget's window changes (or whenever anything else occurs that could change the display in a scrollbar, such as a change in the total size of the widget's contents), the widget will generate a Tcl command by concatenating the scroll command and two numbers. Each of the numbers is a fraction between 0 and 1, which indicates a position in the document. 0 indicates the beginning of the document, 1 indicates the end, .333 indicates a position one third the way through the document, and so on. The first fraction indicates the first information in the document that is visible in the window, and the second fraction indicates the information just after the last portion that is visible. The command is then passed to the Tcl interpreter for execution. Typically the xScrollCommand option consists of the path name of a scrollbar widget followed by “set”, e.g. “.x.scrollbar set”: this will cause the scrollbar to be updated whenever the view in the window changes. If this option is not specified, then no command will be executed.

Command-Line Name: -yscrollcommand
Database Name: yScrollCommand
Database Class: ScrollCommand
Specifies the prefix for a command used to communicate with vertical scrollbars. This option is treated in the same way as the xScrollCommand option, except that it is used for vertical scrollbars and is provided by widgets that support vertical scrolling. See the description of xScrollCommand for details on how this option is used.
SEE ALSO

colors, cursors, font

KEYWORDS

class, name, standard option, switch

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

tk_chooseColor - pops up a dialog box for the user to select a color.

SYNOPSIS

tk_chooseColor ?option value ...?  

DESCRIPTION

The procedure tk_chooseColor pops up a dialog box for the user to select a color. The following option-value pairs are possible as command line arguments:

-initialcolor color  
  Specifies the color to display in the color dialog when it pops up. color must be in a form acceptable to the Tk_GetColor function.

-parent window  
  Makes window the logical parent of the color dialog. The color dialog is displayed on top of its parent window.

-title titleString  
  Specifies a string to display as the title of the dialog box. If this option is not specified, then a default title will be displayed.

If the user selects a color, tk_chooseColor will return the name of the color in a form acceptable to Tk_GetColor. If the user cancels the operation, both commands will return the empty string.

EXAMPLE
button .b -bg [tk_chooseColor -initialcolor gray -ti

KEYWORDS

color selection dialog

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
NAME
text, tk_textCopy, tk_textCut, tk_textPaste - Create and manipulate text widgets

SYNOPSIS

STANDARD OPTIONS
-foreground or -fg, foreground, Foreground
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-exportselection, exportSelection, ExportSelection
-font, font, Font
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-insertbackground, insertBackground, Foreground
-insertborderwidth, insertBorderWidth, BorderWidth
-insertofftime, insertOffTime, OffTime
-insertonetime, insertOnTime, OnTime
-insertwidth, insertWidth, InsertWidth
-padX, padX, Pad
-padY, padY, Pad
-relief, relief, Relief
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-setgrid, setGrid, SetGrid
-takefocus, takeFocus, TakeFocus
-xscrollcommand, xScrollCommand, ScrollCommand
-yscrollcommand, yScrollCommand, ScrollCommand

WIDGET-SPECIFIC OPTIONS
-autoseparators, autoSeparators, AutoSeparators
-blockcursor, blockCursor, BlockCursor
-endline, endLine, EndLine
-height, height, Height
-inactiveselectbackground, inactiveSelectBackground, Foreground
-maxundo, maxUndo, MaxUndo
-spacing1, spacing1, Spacing1
-spacing2, spacing2, Spacing2
-spacing3, spacing3, Spacing3
-startline, startLine, StartLine
-state, state, State
-tabs, tabs, Tabs
-tabstyle, tabStyle, TabStyle
-undo, undo, Undo
-width, width, Width
-wrap, wrap, Wrap

DESCRIPTION

INDICES

line, char
@x, y
end
mark
tag.first
tag.last
pathName
imageName
+ count ?submodifier? chars
- count ?submodifier? chars
+ count ?submodifier? indices
- count ?submodifier? indices
+ count ?submodifier? lines
- count ?submodifier? lines
?submodifier? linestart
?submodifier? lineend
?submodifier? wordstart
?submodifier? wordend
TAGS

- **background** *color*
- **bgstipple** *bitmap*
- **borderwidth** *pixels*
- **elide** *boolean*
- **fgstipple** *bitmap*
- **font** *fontName*
- **foreground** *color*
- **justify** *justify*
- **lmargin1** *pixels*
- **lmargin2** *pixels*
- **offset** *pixels*
- **overstrike** *boolean*
- **relief** *relief*
- **rmargin** *pixels*
- **spacing1** *pixels*
- **spacing2** *pixels*
- **spacing3** *pixels*
- **tabs** *tabList*
- **tabstyle** *style*
- **underline** *boolean*
- **wrap** *mode*

MARKS

EMBEDDED WINDOWS

- **align** *where*
- **create** *script*
- **padx** *pixels*
- **pady** *pixels*
- **stretch** *boolean*
- **window** *pathName*

EMBEDDED IMAGES

- **align** *where*
- **image** *image*
- **name** *ImageName*
- **padx** *pixels*
- **pady** *pixels*

THE SELECTION
THE INSERTION CURSOR
THE MODIFIED FLAG
THE UNDO MECHANISM
PEER WIDGETS
WIDGET COMMAND

`pathName bbox index`
`pathName cget option`
`pathName compare index1 op index2`
`pathName configure ?option? ?value option value ...?`
`pathName count ?options? index1 index2`

- `chars`
- `displaychars`
- `displayindices`
- `displaylines`
- `indices`
- `lines`
- `xpixels`
- `ypixels`

`pathName debug ?boolean?`
`pathName delete index1 ?index2 ...?`
`pathName dlineinfo index`
`pathName dump ?switches? index1 ?index2?`

- `all`
- `command command`
- `image`
- `mark`
- `tag`
- `text`
- `window`

`pathName edit option ?arg arg ...?`
`pathName edit modified ?boolean?`
`pathName edit redo`
`pathName edit reset`
`pathName edit separator`
`pathName edit undo`

`pathName get ?-displaychars? -- index1 ?index2 ...?`
`pathName image option ?arg arg ...?`
pathName **image cget** index option
pathName **image configure** index ?option value ...?
pathName **image create** index ?option value ...?
pathName **image names**

pathName **index index**
pathName **insert** index chars ?tagList chars tagList ...?
pathName **mark option** ?arg arg ...
    pathName **mark gravity** markName ?direction?
    pathName **mark names**
    pathName **mark next** index
    pathName **mark previous** index
    pathName **mark set** markName index
    pathName **mark unset** markName ?markName markName ...?

pathName **peer option** args
    pathName **peer create** newPathName ?options?
    pathName **peer names**

pathName **replace** index1 index2 chars ?tagList chars tagList ...
    pathName **scan option** args
        pathName **scan mark** x y
        pathName **scan dragto** x y
    pathName **search** ?switches? pattern index ?stopIndex?
        -forwards
        -backwards
        -exact
        -regexp
        -nolinestop
        -nocase
        -count varName
        -all
        -overlap
        -strictlimits
        -elide
        --
    pathName **see index**
    pathName **tag option** ?arg arg ...

=pathName tag add tagName index1 ?index2 index1 index2 ...
pathName tag bind tagName ?sequence? ?script?
pathName tag cget tagName option
pathName tag configure tagName ?option? ?value? ? option value ...
pathName tag delete tagName ?tagName ...
pathName tag lower tagName ?belowThis?
pathName tag names ?index?
pathName tag nextrange tagName index1 ?index2?
pathName tag prevrange tagName index1 ?index2?
pathName tag raise tagName ?aboveThis?
pathName tag ranges tagName
pathName tag remove tagName index1 ?index2 index1 index2 ...

pathName window option ?arg arg ...

pathName window cget index option
pathName window configure index ?option value ...
pathName window create index ?option value ...

pathName window names

pathName xview option args
pathName xview
pathName xview moveto fraction
pathName xview scroll number what

pathName yview ?args?

pathName yview
pathName yview moveto fraction
pathName yview scroll number what
pathName yview ?-pickplace? index
pathName yview number

BINDINGS
KNOWN/issues
ISSUES CONCERNING CHARS AND INDICES
PERFORMANCE issues
KNOWN BUGS
SEE ALSO
KEYWORDS
NAME
text, tk_textCopy, tk_textCut, tk_textPaste - Create and manipulate text widgets

SYNOPSIS
text pathName ?options?
tk_textCopy pathName
tk_textCut pathName
tk_textPaste pathName

STANDARD OPTIONS
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-exportselection, exportSelection, ExportSelection
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-insertbackground, insertBackground, Foreground
-insertborderwidth, insertBorderWidth, BorderWidth
-insertofftime, insertOffTime, OffTime
-insertontime, insertOnTime, OnTime
-insertwidth, insertWidth, InsertWidth
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-setgrid, setGrid, SetGrid
-takefocus, takeFocus, TakeFocus
-xscrollcommand, xScrollCommand, ScrollCommand
-yscrollcommand, yScrollCommand, ScrollCommand
**WIDGET-SPECIFIC OPTIONS**

**Command-Line Name:** `-autoseparators`  
**Database Name:** `autoSeparators`  
**Database Class:** `AutoSeparators`  
   Specifies a boolean that says whether separators are automatically inserted in the undo stack. Only meaningful when the `-undo` option is true.

**Command-Line Name:** `-blockcursor`  
**Database Name:** `blockCursor`  
**Database Class:** `BlockCursor`  
   Specifies a boolean that says whether the blinking insertion cursor should be drawn as a character-sized rectangular block. If false (the default) a thin vertical line is used for the insertion cursor.

**Command-Line Name:** `-endline`  
**Database Name:** `endLine`  
**Database Class:** `EndLine`  
   Specifies an integer line index representing the last line of the underlying textual data store that should be contained in the widget. This allows a text widget to reflect only a portion of a larger piece of text. Instead of an integer, the empty string can be provided to this configuration option, which will configure the widget to end at the very last line in the textual data store.

**Command-Line Name:** `-height`  
**Database Name:** `height`  
**Database Class:** `Height`  
   Specifies the desired height for the window, in units of characters in the font given by the `-font` option. Must be at least one.

**Command-Line Name:** `-inactiveselectbackground`  
**Database Name:** `inactiveSelectBackground`  
**Database Class:** `Foreground`  
   Specifies the colour to use for the selection (the `sel` tag) when the window does not have the input focus. If empty, `{}`, then no selection is shown when the window does not have the focus.
Command-Line Name: **-maxundo**
Database Name: **maxUndo**
Database Class: **MaxUndo**
   Specifies the maximum number of compound undo actions on the undo stack. A zero or a negative value imply an unlimited undo stack.

Command-Line Name: **-spacing1**
Database Name: **spacing1**
Database Class: **Spacing1**
   Requests additional space above each text line in the widget, using any of the standard forms for screen distances. If a line wraps, this option only applies to the first line on the display. This option may be overridden with **-spacing1** options in tags.

Command-Line Name: **-spacing2**
Database Name: **spacing2**
Database Class: **Spacing2**
   For lines that wrap (so that they cover more than one line on the display) this option specifies additional space to provide between the display lines that represent a single line of text. The value may have any of the standard forms for screen distances. This option may be overridden with **-spacing2** options in tags.

Command-Line Name: **-spacing3**
Database Name: **spacing3**
Database Class: **Spacing3**
   Requests additional space below each text line in the widget, using any of the standard forms for screen distances. If a line wraps, this option only applies to the last line on the display. This option may be overridden with **-spacing3** options in tags.

Command-Line Name: **-startline**
Database Name: **startLine**
Database Class: **StartLine**
   Specifies an integer line index representing the first line of the underlying textual data store that should be contained in the widget. This allows a text widget to reflect only a portion of a larger
piece of text. Instead of an integer, the empty string can be provided to this configuration option, which will configure the widget to start at the very first line in the textual data store.

Command-Line Name: **-state**  
Database Name: **state**  
Database Class: **State**  
Specifies one of two states for the text: **normal** or **disabled**. If the text is disabled then characters may not be inserted or deleted and no insertion cursor will be displayed, even if the input focus is in the widget.

Command-Line Name: **-tabs**  
Database Name: **tabs**  
Database Class: **Tabs**  
Specifies a set of tab stops for the window. The option's value consists of a list of screen distances giving the positions of the tab stops, each of which is a distance relative to the left edge of the widget (excluding borders, padding, etc). Each position may optionally be followed in the next list element by one of the keywords **left**, **right**, **center**, or **numeric**, which specifies how to justify text relative to the tab stop. **Left** is the default; it causes the text following the tab character to be positioned with its left edge at the tab position. **Right** means that the right edge of the text following the tab character is positioned at the tab position, and **center** means that the text is centered at the tab position. **Numeric** means that the decimal point in the text is positioned at the tab position; if there is no decimal point then the least significant digit of the number is positioned just to the left of the tab position; if there is no number in the text then the text is right-justified at the tab position. For example, “**-tabs {2c left 4c 6c center}**” creates three tab stops at two-centimeter intervals; the first two use left justification and the third uses center justification.

If the list of tab stops does not have enough elements to cover all of the tabs in a text line, then Tk extrapolates new tab stops using the spacing and alignment from the last tab stop in the list. Tab distances must be strictly positive, and must always increase from
one tab stop to the next (if not, an error is thrown). The value of the tabs option may be overridden by -tabs options in tags.

If no -tabs option is specified, or if it is specified as an empty list, then Tk uses default tabs spaced every eight (average size) characters. To achieve a different standard spacing, for example every 4 characters, simply configure the widget with “-tabs "[expr {4 * [font measure $font 0]}] left" -tabstyle wordprocessor”.

Command-Line Name: -tabstyle
Database Name: tabStyle
Database Class: TabStyle
Specifies how to interpret the relationship between tab stops on a line and tabs in the text of that line. The value must be tabular (the default) or wordprocessor. Note that tabs are interpreted as they are encountered in the text. If the tab style is tabular then the \( n \)’th tab character in the line's text will be associated with the \( n \)’th tab stop defined for that line. If the tab character's x coordinate falls to the right of the \( n \)'th tab stop, then a gap of a single space will be inserted as a fallback. If the tab style is wordprocessor then any tab character being laid out will use (and be defined by) the first tab stop to the right of the preceding characters already laid out on that line. The value of the tabstyle option may be overridden by -tabstyle options in tags.

Command-Line Name: -undo
Database Name: undo
Database Class: Undo
Specifies a boolean that says whether the undo mechanism is active or not.

Command-Line Name: -width
Database Name: width
Database Class: Width
Specifies the desired width for the window in units of characters in the font given by the -font option. If the font does not have a uniform width then the width of the character “0” is used in translating from character units to screen units.
Specifies how to handle lines in the text that are too long to be displayed in a single line of the text's window. The value must be none or char or word. A wrap mode of none means that each line of text appears as exactly one line on the screen; extra characters that do not fit on the screen are not displayed. In the other modes each line of text will be broken up into several screen lines if necessary to keep all the characters visible. In char mode a screen line break may occur after any character; in word mode a line break will only be made at word boundaries.

DESCRIPTION

The text command creates a new window (given by the pathName argument) and makes it into a text widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the text such as its default background color and relief. The text command returns the path name of the new window.

A text widget displays one or more lines of text and allows that text to be edited. Text widgets support four different kinds of annotations on the text, called tags, marks, embedded windows or embedded images. Tags allow different portions of the text to be displayed with different fonts and colors. In addition, Tcl commands can be associated with tags so that scripts are invoked when particular actions such as keystrokes and mouse button presses occur in particular ranges of the text. See TAGS below for more details.

The second form of annotation consists of floating markers in the text called “marks”. Marks are used to keep track of various interesting positions in the text as it is edited. See MARKS below for more details.

The third form of annotation allows arbitrary windows to be embedded in a text widget. See EMBEDDED WINDOWS below for more details.

The fourth form of annotation allows Tk images to be embedded in a
text widget. See **EMBEDDED IMAGES** below for more details.

The text widget also has a built-in undo/redo mechanism. See **THE UNDO MECHANISM** below for more details.

The text widget allows for the creation of peer widgets. These are other text widgets which share the same underlying data (text, marks, tags, images, etc). See **PEER WIDGETS** below for more details.

---

### INDICES

Many of the widget commands for texts take one or more indices as arguments. An index is a string used to indicate a particular place within a text, such as a place to insert characters or one endpoint of a range of characters to delete. Indices have the syntax

\[
\text{base} \ 	ext{modifier} \ 	ext{modifier} \ 	ext{modifier} \ldots
\]

Where `base` gives a starting point and the `modifiers` adjust the index from the starting point (e.g. move forward or backward one character). Every index must contain a `base`, but the `modifiers` are optional. Most modifiers (as documented below) allow an optional submodifier. Valid submodifiers are `any` and `display`. If the submodifier is abbreviated, then it must be followed by whitespace, but otherwise there need be no space between the submodifier and the following `modifier`. Typically the `display` submodifier adjusts the meaning of the following `modifier` to make it refer to visual or non-elided units rather than logical units, but this is explained for each relevant case below. Lastly, where `count` is used as part of a modifier, it can be positive or negative, so “`base - -3 lines`” is perfectly valid (and equivalent to “`base +3lines`”).

The `base` for an index must have one of the following forms:

- **line.char**
  Indicates `char`'th character on line `line`. Lines are numbered from 1 for consistency with other UNIX programs that use this numbering scheme. Within a line, characters are numbered from 0. If `char` is
end then it refers to the newline character that ends the line.

@x,y
Indicates the character that covers the pixel whose x and y coordinates within the text's window are x and y.

end
Indicates the end of the text (the character just after the last newline).

mark
Indicates the character just after the mark whose name is mark.

tag.first
Indicates the first character in the text that has been tagged with tag. This form generates an error if no characters are currently tagged with tag.

tag.last
Indicates the character just after the last one in the text that has been tagged with tag. This form generates an error if no characters are currently tagged with tag.

pathName
Indicates the position of the embedded window whose name is pathName. This form generates an error if there is no embedded window by the given name.

imageName
Indicates the position of the embedded image whose name is imageName. This form generates an error if there is no embedded image by the given name.

If the base could match more than one of the above forms, such as a mark and imageName both having the same value, then the form earlier in the above list takes precedence. If modifiers follow the base index, each one of them must have one of the forms listed below. Keywords such as chars and wordend may be abbreviated as long as the
abbreviation is unambiguous.

+ `count ?submodifier? chars`
   Adjust the index forward by `count` characters, moving to later lines in the text if necessary. If there are fewer than `count` characters in the text after the current index, then set the index to the last index in the text. Spaces on either side of `count` are optional. If the `display` submodifier is given, elided characters are skipped over without being counted. If `any` is given, then all characters are counted. For historical reasons, if neither modifier is given then the count actually takes place in units of index positions (see `indices` for details). This behaviour may be changed in a future major release, so if you need an index count, you are encouraged to use `indices` instead wherever possible.

- `count ?submodifier? chars`
   Adjust the index backward by `count` characters, moving to earlier lines in the text if necessary. If there are fewer than `count` characters in the text before the current index, then set the index to the first index in the text (1.0). Spaces on either side of `count` are optional. If the `display` submodifier is given, elided characters are skipped over without being counted. If `any` is given, then all characters are counted. For historical reasons, if neither modifier is given then the count actually takes place in units of index positions (see `indices` for details). This behaviour may be changed in a future major release, so if you need an index count, you are encouraged to use `indices` instead wherever possible.

+ `count ?submodifier? indices`
   Adjust the index forward by `count` index positions, moving to later lines in the text if necessary. If there are fewer than `count` index positions in the text after the current index, then set the index to the last index position in the text. Spaces on either side of `count` are optional. Note that an index position is either a single character or a single embedded image or embedded window. If the `display` submodifier is given, elided indices are skipped over without being counted. If `any` is given, then all indices are counted; this is also the default behaviour if no modifier is given.
- `count ?submodifier? indices`
  Adjust the index backward by `count` index positions, moving to earlier lines in the text if necessary. If there are fewer than `count` index positions in the text before the current index, then set the index to the first index position (1.0) in the text. Spaces on either side of `count` are optional. If the `display` submodifier is given, elided indices are skipped over without being counted. If any is given, then all indices are counted; this is also the default behaviour if no modifier is given.

+ `count ?submodifier? lines`
  Adjust the index forward by `count` lines, retaining the same character position within the line. If there are fewer than `count` lines after the line containing the current index, then set the index to refer to the same character position on the last line of the text. Then, if the line is not long enough to contain a character at the indicated character position, adjust the character position to refer to the last character of the line (the newline). Spaces on either side of `count` are optional. If the `display` submodifier is given, then each visual display line is counted separately. Otherwise, if any (or no modifier) is given, then each logical line (no matter how many times it is visually wrapped) counts just once. If the relevant lines are not wrapped, then these two methods of counting are equivalent.

- `count ?submodifier? lines`
  Adjust the index backward by `count` logical lines, retaining the same character position within the line. If there are fewer than `count` lines before the line containing the current index, then set the index to refer to the same character position on the first line of the text. Then, if the line is not long enough to contain a character at the indicated character position, adjust the character position to refer to the last character of the line (the newline). Spaces on either side of `count` are optional. If the `display` submodifier is given, then each visual display line is counted separately. Otherwise, if any (or no modifier) is given, then each logical line (no matter how many times it is visually wrapped) counts just once. If the relevant lines are not wrapped, then these two methods of counting are equivalent.
equivalent.

?submodifier? linestart
Adjust the index to refer to the first index on the line. If the display submodifier is given, this is the first index on the display line, otherwise on the logical line.

?submodifier? lineend
Adjust the index to refer to the last index on the line (the newline). If the display submodifier is given, this is the last index on the display line, otherwise on the logical line.

?submodifier? wordstart
Adjust the index to refer to the first character of the word containing the current index. A word consists of any number of adjacent characters that are letters, digits, or underscores, or a single character that is not one of these. If the display submodifier is given, this only examines non-elided characters, otherwise all characters (elided or not) are examined.

?submodifier? wordend
Adjust the index to refer to the character just after the last one of the word containing the current index. If the current index refers to the last character of the text then it is not modified. If the display submodifier is given, this only examines non-elided characters, otherwise all characters (elided or not) are examined.

If more than one modifier is present then they are applied in left-to-right order. For example, the index “end - 1 chars” refers to the next-to-last character in the text and “insert wordstart - 1 c” refers to the character just before the first one in the word containing the insertion cursor. Modifiers are applied one by one in this left to right order, and after each step the resulting index is constrained to be a valid index in the text widget. So, for example, the index “1.0 -1c +1c” refers to the index “2.0”.

Where modifiers result in index changes by display lines, display chars or display indices, and the base refers to an index inside an elided tag,
that base index is considered to be equivalent to the first following non-elided index.

TAGS

The first form of annotation in text widgets is a tag. A tag is a textual string that is associated with some of the characters in a text. Tags may contain arbitrary characters, but it is probably best to avoid using the characters “ ” (space), +, or -: these characters have special meaning in indices, so tags containing them cannot be used as indices. There may be any number of tags associated with characters in a text. Each tag may refer to a single character, a range of characters, or several ranges of characters. An individual character may have any number of tags associated with it.

A priority order is defined among tags, and this order is used in implementing some of the tag-related functions described below. When a tag is defined (by associating it with characters or setting its display options or binding commands to it), it is given a priority higher than any existing tag. The priority order of tags may be redefined using the “*pathName tag raise*” and “*pathName tag lower*” widget commands.

Tags serve three purposes in text widgets. First, they control the way information is displayed on the screen. By default, characters are displayed as determined by the -*background*, -*font*, and -*foreground* options for the text widget. However, display options may be associated with individual tags using the “*pathName tag configure*” widget command. If a character has been tagged, then the display options associated with the tag override the default display style. The following options are currently supported for tags:

- **-background** *color*
  
  *Color* specifies the background color to use for characters associated with the tag. It may have any of the forms accepted by *Tk_GetColor*.

- **-bgstipple** *bitmap*
  
  *Bitmap* specifies a bitmap that is used as a stipple pattern for the
background. It may have any of the forms accepted by `Tk_GetBitmap`. If `bitmap` has not been specified, or if it is specified as an empty string, then a solid fill will be used for the background.

**-borderwidth pixels**

`Pixels` specifies the width of a 3-D border to draw around the background. It may have any of the forms accepted by `Tk_GetPixels`. This option is used in conjunction with the `-relief` option to give a 3-D appearance to the background for characters; it is ignored unless the `-background` option has been set for the tag.

**-elide boolean**

`Elide` specifies whether the data should be elided. Elided data (characters, images, embedded windows, etc) is not displayed and takes no space on screen, but further on behaves just as normal data.

**-fgstipple bitmap**

`Bitmap` specifies a bitmap that is used as a stipple pattern when drawing text and other foreground information such as underlines. It may have any of the forms accepted by `Tk_GetBitmap`. If `bitmap` has not been specified, or if it is specified as an empty string, then a solid fill will be used.

**-font fontName**

`fontName` is the name of a font to use for drawing characters. It may have any of the forms accepted by `Tk_GetFont`.

**-foreground color**

`Color` specifies the color to use when drawing text and other foreground information such as underlines. It may have any of the forms accepted by `Tk_GetColor`.

**-justify justify**

If the first non-elided character of a display line has a tag for which this option has been specified, then `justify` determines how to justify the line. It must be one of `left`, `right`, or `center`. If a line wraps, then
the justification for each line on the display is determined by the first non-elided character of that display line.

-**Imargin1** pixels
  If the first non-elided character of a text line has a tag for which this option has been specified, then *pixels* specifies how much the line should be indented from the left edge of the window. *Pixels* may have any of the standard forms for screen distances. If a line of text wraps, this option only applies to the first line on the display; the **Imargin2** option controls the indentation for subsequent lines.

-**Imargin2** pixels
  If the first non-elided character of a display line has a tag for which this option has been specified, and if the display line is not the first for its text line (i.e., the text line has wrapped), then *pixels* specifies how much the line should be indented from the left edge of the window. *Pixels* may have any of the standard forms for screen distances. This option is only used when wrapping is enabled, and it only applies to the second and later display lines for a text line.

-**offset** pixels
  *Pixels* specifies an amount by which the text's baseline should be offset vertically from the baseline of the overall line, in pixels. For example, a positive offset can be used for superscripts and a negative offset can be used for subscripts. *Pixels* may have any of the standard forms for screen distances.

-**overstrike** boolean
  Specifies whether or not to draw a horizontal rule through the middle of characters. *Boolean* may have any of the forms accepted by *Tcl_GetBoolean*.

-**relief** relief
  *Relief* specifies the 3-D relief to use for drawing backgrounds, in any of the forms accepted by *Tk_GetRelief*. This option is used in conjunction with the **-borderwidth** option to give a 3-D appearance to the background for characters; it is ignored unless the **background** option has been set for the tag.
-rmargin pixels
If the first non-elided character of a display line has a tag for which this option has been specified, then pixels specifies how wide a margin to leave between the end of the line and the right edge of the window. Pixels may have any of the standard forms for screen distances. This option is only used when wrapping is enabled. If a text line wraps, the right margin for each line on the display is determined by the first non-elided character of that display line.

-spacing1 pixels
Pixels specifies how much additional space should be left above each text line, using any of the standard forms for screen distances. If a line wraps, this option only applies to the first line on the display.

-spacing2 pixels
For lines that wrap, this option specifies how much additional space to leave between the display lines for a single text line. Pixels may have any of the standard forms for screen distances.

-spacing3 pixels
Pixels specifies how much additional space should be left below each text line, using any of the standard forms for screen distances. If a line wraps, this option only applies to the last line on the display.

-tabs tabList
TabList specifies a set of tab stops in the same form as for the -tabs option for the text widget. This option only applies to a display line if it applies to the first non-elided character on that display line. If this option is specified as an empty string, it cancels the option, leaving it unspecified for the tag (the default). If the option is specified as a non-empty string that is an empty list, such as -tags { }, then it requests default 8-character tabs as described for the -tags widget option.

-tabstyle style
Style specifies either the tabular or wordprocessor style of tabbing
to use for the text widget. This option only applies to a display line if it applies to the first non-elided character on that display line. If this option is specified as an empty string, it cancels the option, leaving it unspecified for the tag (the default).

-underline boolean

Boolean specifies whether or not to draw an underline underneath characters. It may have any of the forms accepted by Tcl_GetBoolean.

-wrap mode

Mode specifies how to handle lines that are wider than the text's window. It has the same legal values as the -wrap option for the text widget: none, char, or word. If this tag option is specified, it overrides the -wrap option for the text widget.

If a character has several tags associated with it, and if their display options conflict, then the options of the highest priority tag are used. If a particular display option has not been specified for a particular tag, or if it is specified as an empty string, then that option will never be used; the next-highest-priority tag's option will used instead. If no tag specifies a particular display option, then the default style for the widget will be used.

The second purpose for tags is event bindings. You can associate bindings with a tag in much the same way you can associate bindings with a widget class: whenever particular X events occur on characters with the given tag, a given Tcl command will be executed. Tag bindings can be used to give behaviors to ranges of characters; among other things, this allows hypertext-like features to be implemented. For details, see the description of the “pathName tag bind” widget command below. Tag bindings are shared between all peer widgets (including any bindings for the special sel tag).

The third use for tags is in managing the selection. See THE SELECTION below. With the exception of the special sel tag, all tags are shared between peer text widgets, and may be manipulated on an equal basis from any such widget. The sel tag exists separately and
independently in each peer text widget (but any tag bindings to \texttt{sel} are shared).

\section*{MARKS}

The second form of annotation in text widgets is a mark. Marks are used for remembering particular places in a text. They are something like tags, in that they have names and they refer to places in the file, but a mark is not associated with particular characters. Instead, a mark is associated with the gap between two characters. Only a single position may be associated with a mark at any given time. If the characters around a mark are deleted the mark will still remain; it will just have new neighbor characters. In contrast, if the characters containing a tag are deleted then the tag will no longer have an association with characters in the file. Marks may be manipulated with the \texttt{pathName mark} widget command, and their current locations may be determined by using the mark name as an index in widget commands.

Each mark also has a “gravity”, which is either \texttt{left} or \texttt{right}. The gravity for a mark specifies what happens to the mark when text is inserted at the point of the mark. If a mark has left gravity, then the mark is treated as if it were attached to the character on its left, so the mark will remain to the left of any text inserted at the mark position. If the mark has right gravity, new text inserted at the mark position will appear to the left of the mark (so that the mark remains rightmost). The gravity for a mark defaults to \texttt{right}.

The name space for marks is different from that for tags: the same name may be used for both a mark and a tag, but they will refer to different things.

Two marks have special significance. First, the mark \texttt{insert} is associated with the insertion cursor, as described under \texttt{THE INSERTION CURSOR} below. Second, the mark \texttt{current} is associated with the character closest to the mouse and is adjusted automatically to track the mouse position and any changes to the text in the widget (one exception: \texttt{current} is not updated in response to mouse motions if a mouse button is down; the update will be deferred until all mouse
buttons have been released). Neither of these special marks may be deleted. With the exception of these two special marks, all marks are shared between peer text widgets, and may be manipulated on an equal basis from any peer.

**EMBEDDED WINDOWS**

The third form of annotation in text widgets is an embedded window. Each embedded window annotation causes a window to be displayed at a particular point in the text. There may be any number of embedded windows in a text widget, and any widget may be used as an embedded window (subject to the usual rules for geometry management, which require the text window to be the parent of the embedded window or a descendant of its parent). The embedded window's position on the screen will be updated as the text is modified or scrolled, and it will be mapped and unmapped as it moves into and out of the visible area of the text widget. Each embedded window occupies one unit's worth of index space in the text widget, and it may be referred to either by the name of its embedded window or by its position in the widget's index space. If the range of text containing the embedded window is deleted then the window is destroyed. Similarly if the text widget as a whole is deleted, then the window is destroyed.

When an embedded window is added to a text widget with the *pathName window create* widget command, several configuration options may be associated with it. These options may be modified later with the *pathName window configure* widget command. The following options are currently supported:

**-align where**

If the window is not as tall as the line in which it is displayed, this option determines where the window is displayed in the line. *Where* must have one of the values **top** (align the top of the window with the top of the line), **center** (center the window within the range of the line), **bottom** (align the bottom of the window with the bottom of the line's area), or **baseline** (align the bottom of the window with the baseline of the line).
-create script
Specifies a Tcl script that may be evaluated to create the window for the annotation. If no -window option has been specified for the annotation this script will be evaluated when the annotation is about to be displayed on the screen. Script must create a window for the annotation and return the name of that window as its result. Two substitutions will be performed in script before evaluation. %W will be substituted by the name of the parent text widget, and %% will be substituted by a single %. If the annotation's window should ever be deleted, script will be evaluated again the next time the annotation is displayed.

-padx pixels
Pixels specifies the amount of extra space to leave on each side of the embedded window. It may have any of the usual forms defined for a screen distance.

-pady pixels
Pixels specifies the amount of extra space to leave on the top and on the bottom of the embedded window. It may have any of the usual forms defined for a screen distance.

-stretch boolean
If the requested height of the embedded window is less than the height of the line in which it is displayed, this option can be used to specify whether the window should be stretched vertically to fill its line. If the -pady option has been specified as well, then the requested padding will be retained even if the window is stretched.

-window pathName
Specifies the name of a window to display in the annotation. Note that if a pathName has been set, then later configuring a window to the empty string will not delete the widget corresponding to the old pathName. Rather it will remove the association between the old pathName and the text widget. If multiple peer widgets are in use, it is usually simpler to use the -create option if embedded windows are desired in each peer.
The final form of annotation in text widgets is an embedded image. Each embedded image annotation causes an image to be displayed at a particular point in the text. There may be any number of embedded images in a text widget, and a particular image may be embedded in multiple places in the same text widget. The embedded image's position on the screen will be updated as the text is modified or scrolled. Each embedded image occupies one unit's worth of index space in the text widget, and it may be referred to either by its position in the widget's index space, or the name it is assigned when the image is inserted into the text widget with `pathName image create`. If the range of text containing the embedded image is deleted then that copy of the image is removed from the screen.

When an embedded image is added to a text widget with the `pathName image create` widget command, a name unique to this instance of the image is returned. This name may then be used to refer to this image instance. The name is taken to be the value of the `-name` option (described below). If the `-name` option is not provided, the `-image` name is used instead. If the `imageName` is already in use in the text widget, then `#nn` is added to the end of the `imageName`, where `nn` is an arbitrary integer. This insures the `imageName` is unique. Once this name is assigned to this instance of the image, it does not change, even though the `-image` or `-name` values can be changed with `pathName image configure`.

When an embedded image is added to a text widget with the `pathName image create` widget command, several configuration options may be associated with it. These options may be modified later with the `pathName image configure` widget command. The following options are currently supported:

- **-align where**
  If the image is not as tall as the line in which it is displayed, this option determines where the image is displayed in the line. Where must have one of the values `top` (align the top of the image with the top of the line), `center` (center the image within the range of the
line), **bottom** (align the bottom of the image with the bottom of the line’s area), or **baseline** (align the bottom of the image with the baseline of the line).

**-image image**
Specifies the name of the Tk image to display in the annotation. If *image* is not a valid Tk image, then an error is returned.

**-name ImageName**
Specifies the name by which this image instance may be referenced in the text widget. If *ImageName* is not supplied, then the name of the Tk image is used instead. If the *imageName* is already in use, *#nn* is appended to the end of the name as described above.

**-padx pixels**
*Pixels* specifies the amount of extra space to leave on each side of the embedded image. It may have any of the usual forms defined for a screen distance.

**-pady pixels**
*Pixels* specifies the amount of extra space to leave on the top and on the bottom of the embedded image. It may have any of the usual forms defined for a screen distance.

---

### THE SELECTION

Selection support is implemented via tags. If the **exportSelection** option for the text widget is true then the **sel** tag will be associated with the selection:

[1] Whenever characters are tagged with **sel** the text widget will claim ownership of the selection.

[2] Attempts to retrieve the selection will be serviced by the text widget, returning all the characters with the **sel** tag.
If the selection is claimed away by another application or by another window within this application, then the sel tag will be removed from all characters in the text.

Whenever the sel tag range changes a virtual event <<Selection>> is generated.

The sel tag is automatically defined when a text widget is created, and it may not be deleted with the “pathName tag delete” widget command. Furthermore, the selectBackground, selectBorderWidth, and selectForeground options for the text widget are tied to the -background, -borderwidth, and -foreground options for the sel tag: changes in either will automatically be reflected in the other. Also the -inactiveselectbackground option for the text widget is used instead of -selectbackground when the text widget does not have the focus. This allows programmatic control over the visualization of the sel tag for foreground and background windows, or to have sel not shown at all (when -inactiveselectbackground is empty) for background windows. Each peer text widget has its own sel tag which can be separately configured and set.

THE INSERTION CURSOR

The mark named insert has special significance in text widgets. It is defined automatically when a text widget is created and it may not be unset with the “pathName mark unset” widget command. The insert mark represents the position of the insertion cursor, and the insertion cursor will automatically be drawn at this point whenever the text widget has the input focus.

THE MODIFIED FLAG

The text widget can keep track of changes to the content of the widget by means of the modified flag. Inserting or deleting text will set this flag. The flag can be queried, set and cleared programatically as well. Whenever the flag changes state a <<Modified>> virtual event is
THE UNDO MECHANISM

The text widget has an unlimited undo and redo mechanism (when the -undo widget option is true) which records every insert and delete action on a stack.

Boundaries (called “separators”) are inserted between edit actions. The purpose of these separators is to group inserts, deletes and replaces into one compound edit action. When undoing a change everything between two separators will be undone. The undone changes are then moved to the redo stack, so that an undone edit can be redone again. The redo stack is cleared whenever new edit actions are recorded on the undo stack. The undo and redo stacks can be cleared to keep their depth under control.

Separators are inserted automatically when the -autoseparators widget option is true. You can insert separators programmatically as well. If a separator is already present at the top of the undo stack no other will be inserted. That means that two separators on the undo stack are always separated by at least one insert or delete action.

The undo mechanism is also linked to the modified flag. This means that undoing or redoing changes can take a modified text widget back to the unmodified state or vice versa. The modified flag will be set automatically to the appropriate state. This automatic coupling does not work when the modified flag has been set by the user, until the flag has been reset again.

See below for the pathName edit widget command that controls the undo mechanism.

PEER WIDGETS

The text widget has a separate store of all its data concerning each line’s textual contents, marks, tags, images and windows, and the undo
While this data store cannot be accessed directly (i.e. without a text widget as an intermediary), multiple text widgets can be created, each of which present different views on the same underlying data. Such text widgets are known as peer text widgets.

As text is added, deleted, edited and coloured in any one widget, and as images, marks, tags are adjusted, all such changes will be reflected in all peers.

All data and markup is shared, except for a few small details. First, the `<sel>` tag may be set and configured (in its display style) differently for each peer. Second, each peer has its own `insert` and `current` mark positions (but all other marks are shared). Third, embedded windows, which are arbitrary other widgets, cannot be shared between peers. This means the `-window` option of embedded windows is independently set for each peer (it is advisable to use the `-create` script capabilities to allow each peer to create its own embedded windows as needed). Fourth, all of the configuration options of each peer (e.g. `-font`, etc) can be set independently, with the exception of `-undo`, `-maxUndo`, `-autoSeparators` (i.e. all undo, redo and modified state issues are shared).

Finally any single peer need not contain all lines from the underlying data store. When creating a peer, a contiguous range of lines (e.g. only lines 52 through 125) may be specified. This allows a peer to contain just a small portion of the overall text. The range of lines will expand and contract as text is inserted or deleted. The peer will only ever display complete lines of text (one cannot share just part of a line). If the peer's contents contracts to nothing (i.e. all complete lines in the peer widget have been deleted from another widget), then it is impossible for new lines to be inserted. The peer will simply become an empty shell on which the background can be configured, but which will never show any content (without manual reconfiguration of the start and end lines). Note that a peer which does not contain all of the underlying data store still has indices numbered from "1.0" to "end". It is simply that those indices reflect a subset of the total data, and data outside the
contained range is not accessible to the peer. This means that the command *peerName index end* may return quite different values in different peers. Similarly, commands like *peerName tag ranges* will not return index ranges outside that which is meaningful to the peer. The configuration options *-startline* and *-endline* may be used to control how much of the underlying data is contained in any given text widget.

Note that peers are really peers. Deleting the “original” text widget will not cause any other peers to be deleted, or otherwise affected.

See below for the *pathname peer* widget command that controls the creation of peer widgets.

---

**WIDGET COMMAND**

The text command creates a new Tcl command whose name is the same as the path name of the text's window. This command may be used to invoke various operations on the widget. It has the following general form:

```text
classoption ?arg arg ...?
```

*PathName* is the name of the command, which is the same as the text widget's path name. *Option* and the *args* determine the exact behavior of the command. The following commands are possible for text widgets:

**pathname bbox index**

Returns a list of four elements describing the screen area of the character given by *index*. The first two elements of the list give the x and y coordinates of the upper-left corner of the area occupied by the character, and the last two elements give the width and height of the area. If the character is only partially visible on the screen, then the return value reflects just the visible part. If the character is not visible on the screen then the return value is an empty list.

**pathname cget option**

Returns the current value of the configuration option given by
option. Option may have any of the values accepted by the text command.

`pathName compare index1 op index2`

Compares the indices given by `index1` and `index2` according to the relational operator given by `op`, and returns 1 if the relationship is satisfied and 0 if it is not. `Op` must be one of the operators `<`, `<=`, `==`, `>=`, `>`, or `!=`. If `op` is `==` then 1 is returned if the two indices refer to the same character, if `op` is `<` then 1 is returned if `index1` refers to an earlier character in the text than `index2`, and so on.

`pathName configure ?option? ?value option value ...?`

Query or modify the configuration options of the widget. If no `option` is specified, returns a list describing all of the available options for `pathName` (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option-value` pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. `Option` may have any of the values accepted by the `text` command.

`pathName count ?options? index1 index2`

Counts the number of relevant things between the two indices. If `index1` is after `index2`, the result will be a negative number (and this holds for each of the possible options). The actual items which are counted depend on the options given. The result is a list of integers, one for the result of each counting option given. Valid counting options are `-chars`, `-displaychars`, `-displayindices`, `-displaylines`, `-indices`, `-lines`, `-xpixels` and `-ypixels`. The default value, if no option is specified, is `-indices`. There is an additional possible option `-update` which is a modifier. If given, then all subsequent options ensure that any possible out of date information is recalculated. This currently only has any effect for the `-ypixels` count (which, if `-update` is not given, will use the text widget's current cached value for each line). The count options are interpreted as follows:
-chars
   count all characters, whether elided or not. Do not count embedded windows or images.

-displaychars
   count all non-elided characters.

-displayindices
   count all non-elided characters, windows and images.

-displaylines
   count all display lines (i.e. counting one for each time a line wraps) from the line of the first index up to, but not including the display line of the second index. Therefore if they are both on the same display line, zero will be returned. By definition displaylines are visible and therefore this only counts portions of actual visible lines.

-indices
   count all characters and embedded windows or images (i.e. everything which counts in text-widget index space), whether they are elided or not.

-lines
   count all logical lines (irrespective of wrapping) from the line of the first index up to, but not including the line of the second index. Therefore if they are both on the same line, zero will be returned. Logical lines are counted whether they are currently visible (non-elided) or not.

-xpixels
   count the number of horizontal pixels from the first pixel of the first index to (but not including) the first pixel of the second index. To count the total desired width of the text widget (assuming wrapping is not enabled), first find the longest line and then use ".text count -xpixels "${line}.0" "${line}.0 lineend"."
ypixels

count the number of vertical pixels from the first pixel of the first index to (but not including) the first pixel of the second index. If both indices are on the same display line, zero will be returned. To count the total number of vertical pixels in the text widget, use “.text count -ypixels 1.0 end”, and to ensure this is up to date, use “.text count -update -ypixels 1.0 end”.

The command returns a positive or negative integer corresponding to the number of items counted between the two indices. One such integer is returned for each counting option given, so a list is returned if more than one option was supplied. For example “.text count -xpixels -ypixels 1.3 4.5” is perfectly valid and will return a list of two elements.

**pathName debug ?boolean?**

If **boolean** is specified, then it must have one of the true or false values accepted by **Tcl_GetBoolean**. If the value is a true one then internal consistency checks will be turned on in the B-tree code associated with text widgets. If **boolean** has a false value then the debugging checks will be turned off. In either case the command returns an empty string. If **boolean** is not specified then the command returns **on** or **off** to indicate whether or not debugging is turned on. There is a single debugging switch shared by all text widgets: turning debugging on or off in any widget turns it on or off for all widgets. For widgets with large amounts of text, the consistency checks may cause a noticeable slow-down.

When debugging is turned on, the drawing routines of the text widget set the global variables **tk_textRedraw** and **tk_textRelayout** to the lists of indices that are redrawn. The values of these variables are tested by Tk's test suite.

**pathName delete index1 ?index2 ...?**

Delete a range of characters from the text. If both **index1** and **index2** are specified, then delete all the characters starting with the one given by **index1** and stopping just before **index2** (i.e. the character at **index2** is not deleted). If **index2** does not specify a
position later in the text than \textit{index1} then no characters are deleted. If \textit{index2} is not specified then the single character at \textit{index1} is deleted. It is not allowable to delete characters in a way that would leave the text without a newline as the last character. The command returns an empty string. If more indices are given, multiple ranges of text will be deleted. All indices are first checked for validity before any deletions are made. They are sorted and the text is removed from the last range to the first range to deleted text does not cause an undesired index shifting side-effects. If multiple ranges with the same start index are given, then the longest range is used. If overlapping ranges are given, then they will be merged into spans that do not cause deletion of text outside the given ranges due to text shifted during deletion.

\textbf{pathName dlineinfo index}

Returns a list with five elements describing the area occupied by the display line containing \textit{index}. The first two elements of the list give the x and y coordinates of the upper-left corner of the area occupied by the line, the third and fourth elements give the width and height of the area, and the fifth element gives the position of the baseline for the line, measured down from the top of the area. All of this information is measured in pixels. If the current wrap mode is \texttt{none} and the line extends beyond the boundaries of the window, the area returned reflects the entire area of the line, including the portions that are out of the window. If the line is shorter than the full width of the window then the area returned reflects just the portion of the line that is occupied by characters and embedded windows. If the display line containing \textit{index} is not visible on the screen then the return value is an empty list.

\textbf{pathName dump \text{"switches\text{"? \text{"index1 \text{"? \text{"index2\text{"?\text{"\

Return the contents of the text widget from \textit{index1} up to, but not including \textit{index2}, including the text and information about marks, tags, and embedded windows. If \textit{index2} is not specified, then it defaults to one character past \textit{index1}. The information is returned in the following format:

\texttt{key1 value1 index1 key2 value2 index2 ...}
The possible key values are **text**, **mark**, **tagon**, **tagoff**, **image**, and **window**. The corresponding value is the text, mark name, tag name, image name, or window name. The index information is the index of the start of the text, mark, tag transition, image or window. One or more of the following switches (or abbreviations thereof) may be specified to control the dump:

- **-all**
  Return information about all elements: text, marks, tags, images and windows. This is the default.

- **-command command**
  Instead of returning the information as the result of the dump operation, invoke the **command** on each element of the text widget within the range. The command has three arguments appended to it before it is evaluated: the **key**, **value**, and **index**.

- **-image**
  Include information about images in the dump results.

- **-mark**
  Include information about marks in the dump results.

- **-tag**
  Include information about tag transitions in the dump results. Tag information is returned as **tagon** and **tagoff** elements that indicate the begin and end of each range of each tag, respectively.

- **-text**
  Include information about text in the dump results. The value is the text up to the next element or the end of range indicated by **index2**. A text element does not span newlines. A multi-line block of text that contains no marks or tag transitions will still be dumped as a set of text segments that each end with a newline. The newline is part of the value.
-window
Include information about embedded windows in the dump results. The value of a window is its Tk pathname, unless the window has not been created yet. (It must have a create script.) In this case an empty string is returned, and you must query the window by its index position to get more information.

`pathName edit option ?arg arg ...?`
This command controls the undo mechanism and the modified flag. The exact behavior of the command depends on the `option` argument that follows the `edit` argument. The following forms of the command are currently supported:

`pathName edit modified ?boolean?`
If `boolean` is not specified, returns the modified flag of the widget. The insert, delete, edit undo and edit redo commands or the user can set or clear the modified flag. If `boolean` is specified, sets the modified flag of the widget to `boolean`.

`pathName edit redo`
When the `-undo` option is true, reapplies the last undone edits provided no other edits were done since then. Generates an error when the redo stack is empty. Does nothing when the `-undo` option is false.

`pathName edit reset`
Clears the undo and redo stacks.

`pathName edit separator`
Inserts a separator (boundary) on the undo stack. Does nothing when the `-undo` option is false.

`pathName edit undo`
Undoes the last edit action when the `-undo` option is true. An edit action is defined as all the insert and delete commands that are recorded on the undo stack in between two separators. Generates an error when the undo stack is empty. Does nothing when the `-undo` option is false.
**pathName get** ?-displaychars? -- index1 ?index2 ...?

Return a range of characters from the text. The return value will be all the characters in the text starting with the one whose index is `index1` and ending just before the one whose index is `index2` (the character at `index2` will not be returned). If `index2` is omitted then the single character at `index1` is returned. If there are no characters in the specified range (e.g. `index1` is past the end of the file or `index2` is less than or equal to `index1`) then an empty string is returned. If the specified range contains embedded windows, no information about them is included in the returned string. If multiple index pairs are given, multiple ranges of text will be returned in a list. Invalid ranges will not be represented with empty strings in the list. The ranges are returned in the order passed to **pathName get**. If the **-displaychars** option is given, then, within each range, only those characters which are not elided will be returned. This may have the effect that some of the returned ranges are empty strings.

**pathName image option** ?arg arg ...?

This command is used to manipulate embedded images. The behavior of the command depends on the **option** argument that follows the **tag** argument. The following forms of the command are currently supported:

**pathName image cget index option**

Returns the value of a configuration option for an embedded image. **Index** identifies the embedded image, and **option** specifies a particular configuration option, which must be one of the ones listed in the section **EMBEDDED IMAGES**.

**pathName image configure index** ?option value ...

Query or modify the configuration options for an embedded image. If no **option** is specified, returns a list describing all of the available options for the embedded image at **index** (see **Tk_ConfigureInfo** for information on the format of this list). If **option** is specified with no **value**, then the command returns a list describing the one named **option** (this list will be identical to the corresponding sublist of the value returned if no **option** is specified). If one or more **option-value** pairs are specified, then
the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. See EMBEDDED IMAGES for information on the options that are supported.

**pathName image create index ?option value ...?**
This command creates a new image annotation, which will appear in the text at the position given by `index`. Any number of option-value pairs may be specified to configure the annotation. Returns a unique identifier that may be used as an index to refer to this image. See EMBEDDED IMAGES for information on the options that are supported, and a description of the identifier returned.

**pathName image names**
Returns a list whose elements are the names of all image instances currently embedded in `window`.

**pathName index index**
Returns the position corresponding to `index` in the form `line.char` where `line` is the line number and `char` is the character number. `Index` may have any of the forms described under INDICES above.

**pathName insert index chars ?tagList chars tagList ...?**
Inserts all of the `chars` arguments just before the character at `index`. If `index` refers to the end of the text (the character after the last newline) then the new text is inserted just before the last newline instead. If there is a single `chars` argument and no `tagList`, then the new text will receive any tags that are present on both the character before and the character after the insertion point; if a tag is present on only one of these characters then it will not be applied to the new text. If `tagList` is specified then it consists of a list of tag names; the new characters will receive all of the tags in this list and no others, regardless of the tags present around the insertion point. If multiple `chars-tagList` argument pairs are present, they produce the same effect as if a separate `pathName insert` widget command had been issued for each pair, in order. The last `tagList` argument may be omitted.
**pathName** **mark** **option** ?arg arg ...?

This command is used to manipulate marks. The exact behavior of the command depends on the **option** argument that follows the **mark** argument. The following forms of the command are currently supported:

**pathName** **mark gravity** **markName** ?direction?

If **direction** is not specified, returns **left** or **right** to indicate which of its adjacent characters **markName** is attached to. If **direction** is specified, it must be **left** or **right**; the gravity of **markName** is set to the given value.

**pathName** **mark names**

Returns a list whose elements are the names of all the marks that are currently set.

**pathName** **mark next index**

Returns the name of the next mark at or after **index**. If **index** is specified in numerical form, then the search for the next mark begins at that index. If **index** is the name of a mark, then the search for the next mark begins immediately after that mark. This can still return a mark at the same position if there are multiple marks at the same index. These semantics mean that the **mark next** operation can be used to step through all the marks in a text widget in the same order as the mark information returned by the **pathName dump** operation. If a mark has been set to the special **end** index, then it appears to be **after end** with respect to the **pathName mark next** operation. An empty string is returned if there are no marks after **index**.

**pathName** **mark previous index**

Returns the name of the mark at or before **index**. If **index** is specified in numerical form, then the search for the previous mark begins with the character just before that index. If **index** is the name of a mark, then the search for the next mark begins immediately before that mark. This can still return a mark at the same position if there are multiple marks at the
same index. These semantics mean that the `pathName mark previous` operation can be used to step through all the marks in a text widget in the reverse order as the mark information returned by the `pathName dump` operation. An empty string is returned if there are no marks before `index`.

`pathName mark set markName index`
Sets the mark named `markName` to a position just before the character at `index`. If `markName` already exists, it is moved from its old position; if it does not exist, a new mark is created. This command returns an empty string.

`pathName mark unset markName ?markName markName ...?`
Remove the mark corresponding to each of the `markName` arguments. The removed marks will not be usable in indices and will not be returned by future calls to “`pathName mark names`”. This command returns an empty string.

`pathName peer option args`  
This command is used to create and query widget peers. It has two forms, depending on `option`:

`pathName peer create newPathName ?options?`  
Creates a peer text widget with the given `newPathName`, and any optional standard configuration options (as for the `text` command). By default the peer will have the same start and end line as the parent widget, but these can be overridden with the standard configuration options.

`pathName peer names`  
Returns a list of peers of this widget (this does not include the widget itself). The order within this list is undefined.

`pathName replace index1 index2 chars ?tagList chars tagList ...?`  
Replaces the range of characters between `index1` and `index2` with the given characters and tags. See the section on `pathName insert` for an explanation of the handling of the `tagList...` arguments, and the section on `pathName delete` for an explanation of the handling
of the indices. If \textit{index2} corresponds to an index earlier in the text than \textit{index1}, an error will be generated.

The deletion and insertion are arranged so that no unnecessary scrolling of the window or movement of insertion cursor occurs. In addition the undo/redo stack are correctly modified, if undo operations are active in the text widget. The command returns an empty string.

\textit{pathName scan option args}
This command is used to implement scanning on texts. It has two forms, depending on \textit{option}:

\textit{pathName scan mark x y}
Records \(x\) and \(y\) and the current view in the text window, for use in conjunction with later \textit{pathName scan dragto} commands. Typically this command is associated with a mouse button press in the widget. It returns an empty string.

\textit{pathName scan dragto x y}
This command computes the difference between its \(x\) and \(y\) arguments and the \(x\) and \(y\) arguments to the last \textit{pathName scan mark} command for the widget. It then adjusts the view by 10 times the difference in coordinates. This command is typically associated with mouse motion events in the widget, to produce the effect of dragging the text at high speed through the window. The return value is an empty string.

\textit{pathName search ?switches? pattern index ?stopIndex?}
Searches the text in \textit{pathName} starting at \textit{index} for a range of characters that matches \textit{pattern}. If a match is found, the index of the first character in the match is returned as result; otherwise an empty string is returned. One or more of the following switches (or abbreviations thereof) may be specified to control the search:

\textbf{-forwards}
The search will proceed forward through the text, finding the first matching range starting at or after the position given by
index. This is the default.

-backwards
The search will proceed backward through the text, finding the matching range closest to index whose first character is before index (it is not allowed to be at index). Note that, for a variety of reasons, backwards searches can be substantially slower than forwards searches (particularly when using -regexp), so it is recommended that performance-critical code use forward searches.

-exact
Use exact matching: the characters in the matching range must be identical to those in pattern. This is the default.

-regexp
Treat pattern as a regular expression and match it against the text using the rules for regular expressions (see the regexp command for details). The default matching automatically passes both the -lineanchor and -linestop options to the regexp engine (unless -nolinestop is used), so that ^$ match beginning and end of line, and ., [^ sequences will never match the newline character \n.

-nolinestop
This allows . and [^ sequences to match the newline character \n, which they will otherwise not do (see the regexp command for details). This option is only meaningful if -regexp is also given, and an error will be thrown otherwise. For example, to match the entire text, use “pathName search -nolinestop -regexp ".*" 1.0”.

-nocase
Ignore case differences between the pattern and the text.

-count varName
The argument following -count gives the name of a variable; if a match is found, the number of index positions between
beginning and end of the matching range will be stored in the variable. If there are no embedded images or windows in the matching range (and there are no elided characters if -elide is not given), this is equivalent to the number of characters matched. In either case, the range \texttt{matchIdx} to \texttt{matchIdx} + $\texttt{count chars}$ will return the entire matched text.

-\texttt{all}

Find all matches in the given range and return a list of the indices of the first character of each match. If a \texttt{-count varName} switch is given, then \texttt{varName} is also set to a list containing one element for each successful match. Note that, even for exact searches, the elements of this list may be different, if there are embedded images, windows or hidden text. Searches with -\texttt{all} behave very similarly to the Tcl command \texttt{regexp -all}, in that overlapping matches are not normally returned. For example, applying an -\texttt{all} search of the pattern \texttt{“w+”} against \texttt{“hello there”} will just match twice, once for each word, and matching \texttt{“Z[a-z]+Z”} against \texttt{“ZooZooZoo”} will just match once.

-\texttt{overlap}

When performing -\texttt{all} searches, the normal behaviour is that matches which overlap an already-found match will not be returned. This switch changes that behaviour so that all matches which are not totally enclosed within another match are returned. For example, applying an -\texttt{overlap} search of the pattern \texttt{“w+”} against \texttt{“hello there”} will just match twice (i.e. no different to just -\texttt{all}), but matching \texttt{“Z[a-z]+Z”} against \texttt{“ZooZooZoo”} will now match twice. An error will be thrown if this switch is used without -\texttt{all}.

-\texttt{strictlimits}

When performing any search, the normal behaviour is that the start and stop limits are checked with respect to the start of the matching text. With the -\texttt{strictlimits} flag, the entire matching range must lie inside the start and stop limits specified for the match to be valid.
-elide
Find elided (hidden) text as well. By default only displayed text is searched.

--
This switch has no effect except to terminate the list of switches: the next argument will be treated as pattern even if it starts with -.

The matching range may be within a single line of text, or run across multiple lines (if parts of the pattern can match a new-line). For regular expression matching one can use the various newline-matching features such as $ to match the end of a line, ^ to match the beginning of a line, and to control whether . is allowed to match a new-line. If stopIndex is specified, the search stops at that index: for forward searches, no match at or after stopIndex will be considered; for backward searches, no match earlier in the text than stopIndex will be considered. If stopIndex is omitted, the entire text will be searched: when the beginning or end of the text is reached, the search continues at the other end until the starting location is reached again; if stopIndex is specified, no wrap-around will occur. This means that, for example, if the search is -forwards but stopIndex is earlier in the text than startIndex, nothing will ever be found. See KNOWN BUGS below for a number of minor limitations of the pathName search command.

pathName see index
Adjusts the view in the window so that the character given by index is completely visible. If index is already visible then the command does nothing. If index is a short distance out of view, the command adjusts the view just enough to make index visible at the edge of the window. If index is far out of view, then the command centers index in the window.

pathName tag option ?arg arg ...?
This command is used to manipulate tags. The exact behavior of the command depends on the option argument that follows the tag
argument. The following forms of the command are currently supported:

\texttt{pathName tag add tagName index1 \textcolor{red}{index2} index1 index2 ...?}

Associate the tag \textit{tagName} with all of the characters starting with \textit{index1} and ending just before \textit{index2} (the character at \textit{index2} is not tagged). A single command may contain any number of \textit{index1-index2} pairs. If the last \textit{index2} is omitted then the single character at \textit{index1} is tagged. If there are no characters in the specified range (e.g. \textit{index1} is past the end of the file or \textit{index2} is less than or equal to \textit{index1}) then the command has no effect.

\texttt{pathName tag bind tagName \textcolor{red}{sequence}? \textcolor{red}{script}?}

This command associates \textit{script} with the tag given by \textit{tagName}. Whenever the event sequence given by \textit{sequence} occurs for a character that has been tagged with \textit{tagName}, the script will be invoked. This widget command is similar to the \texttt{bind} command except that it operates on characters in a text rather than entire widgets. See the \texttt{bind} manual entry for complete details on the syntax of \textit{sequence} and the substitutions performed on \textit{script} before invoking it. If all arguments are specified then a new binding is created, replacing any existing binding for the same \textit{sequence} and \textit{tagName} (if the first character of \textit{script} is “+” then \textit{script} augments an existing binding rather than replacing it). In this case the return value is an empty string. If \textit{script} is omitted then the command returns the \textit{script} associated with \textit{tagName} and \textit{sequence} (an error occurs if there is no such binding). If both \textit{script} and \textit{sequence} are omitted then the command returns a list of all the sequences for which bindings have been defined for \textit{tagName}.

The only events for which bindings may be specified are those related to the mouse and keyboard (such as \texttt{Enter, Leave, ButtonPress, Motion, and KeyPress}) or virtual events. Event bindings for a text widget use the \texttt{current} mark described under \texttt{MARKS} above. An \texttt{Enter} event triggers for a tag when
the tag first becomes present on the current character, and a **Leave** event triggers for a tag when it ceases to be present on the current character. **Enter** and **Leave** events can happen either because the current mark moved or because the character at that position changed. Note that these events are different than **Enter** and **Leave** events for windows. Mouse and keyboard events are directed to the current character. If a virtual event is used in a binding, that binding can trigger only if the virtual event is defined by an underlying mouse-related or keyboard-related event.

It is possible for the current character to have multiple tags, and for each of them to have a binding for a particular event sequence. When this occurs, one binding is invoked for each tag, in order from lowest-priority to highest priority. If there are multiple matching bindings for a single tag, then the most specific binding is chosen (see the manual entry for the **bind** command for details). **continue** and **break** commands within binding scripts are processed in the same way as for bindings created with the **bind** command.

If bindings are created for the widget as a whole using the **bind** command, then those bindings will supplement the tag bindings. The tag bindings will be invoked first, followed by bindings for the window as a whole.

**pathName tag cget tagName option**

This command returns the current value of the option named **option** associated with the tag given by **tagName**. **Option** may have any of the values accepted by the **pathName tag configure** widget command.

**pathName tag configure tagName ?option? ?value? ?option value ...?**

This command is similar to the **pathName configure** widget command except that it modifies options associated with the tag given by **tagName** instead of modifying options for the overall text widget. If no **option** is specified, the command
returns a list describing all of the available options for *tagName* (see *Tk_ConfigureInfo* for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option-value* pairs are specified, then the command modifies the given option(s) to have the given value(s) in *tagName*; in this case the command returns an empty string. See **TAGS** above for details on the options available for tags.

\[\text{pathName tag delete tagName ?tagName ...?}\]

Deletes all tag information for each of the *tagName* arguments. The command removes the tags from all characters in the file and also deletes any other information associated with the tags, such as bindings and display information. The command returns an empty string.

\[\text{pathName tag lower tagName ?belowThis?}\]

Changes the priority of tag *tagName* so that it is just lower in priority than the tag whose name is *belowThis*. If *belowThis* is omitted, then *tagName*'s priority is changed to make it lowest priority of all tags.

\[\text{pathName tag names ?index?}\]

Returns a list whose elements are the names of all the tags that are active at the character position given by *index*. If *index* is omitted, then the return value will describe all of the tags that exist for the text (this includes all tags that have been named in a "*pathName tag" widget command but have not been deleted by a "*pathName tag delete" widget command, even if no characters are currently marked with the tag). The list will be sorted in order from lowest priority to highest priority.

\[\text{pathName tag nextrange tagName index1 ?index2?}\]

This command searches the text for a range of characters tagged with *tagName* where the first character of the range is no earlier than the character at *index1* and no later than the
character just before \textit{index2} (a range starting at \textit{index2} will not be considered). If several matching ranges exist, the first one is chosen. The command's return value is a list containing two elements, which are the index of the first character of the range and the index of the character just after the last one in the range. If no matching range is found then the return value is an empty string. If \textit{index2} is not given then it defaults to the end of the text.

\textit{pathName tag prevrange} \textit{tagName index1 ?index2?}

This command searches the text for a range of characters tagged with \textit{tagName} where the first character of the range is before the character at \textit{index1} and no earlier than the character at \textit{index2} (a range starting at \textit{index2} will be considered). If several matching ranges exist, the one closest to \textit{index1} is chosen. The command's return value is a list containing two elements, which are the index of the first character of the range and the index of the character just after the last one in the range. If no matching range is found then the return value is an empty string. If \textit{index2} is not given then it defaults to the beginning of the text.

\textit{pathName tag raise} \textit{tagName ?aboveThis?}

Changes the priority of tag \textit{tagName} so that it is just higher in priority than the tag whose name is \textit{aboveThis}. If \textit{aboveThis} is omitted, then \textit{tagName}'s priority is changed to make it highest priority of all tags.

\textit{pathName tag ranges} \textit{tagName}

Returns a list describing all of the ranges of text that have been tagged with \textit{tagName}. The first two elements of the list describe the first tagged range in the text, the next two elements describe the second range, and so on. The first element of each pair contains the index of the first character of the range, and the second element of the pair contains the index of the character just after the last one in the range. If there are no characters tagged with \textit{tag} then an empty string is returned.
**pathname tag remove** *tagName* *index1* ?*index2* *index1* *index2* ...
Remove the tag *tagName* from all of the characters starting at *index1* and ending just before *index2* (the character at *index2* is not affected). A single command may contain any number of *index1*-*index2* pairs. If the last *index2* is omitted then the tag is removed from the single character at *index1*. If there are no characters in the specified range (e.g. *index1* is past the end of the file or *index2* is less than or equal to *index1*) then the command has no effect. This command returns an empty string.

**pathname window option** ?*arg* *arg* ...
This command is used to manipulate embedded windows. The behavior of the command depends on the *option* argument that follows the *tag* argument. The following forms of the command are currently supported:

**pathname window cget** *index* *option*
Returns the value of a configuration option for an embedded window. *Index* identifies the embedded window, and *option* specifies a particular configuration option, which must be one of the ones listed in the section EMBEDDED WINDOWS.

**pathname window configure** *index* ?*option* *value* ...
Query or modify the configuration options for an embedded window. If no *option* is specified, returns a list describing all of the available options for the embedded window at *index* (see Tk_ConfigureInfo for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named *option* (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option-value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. See EMBEDDED WINDOWS for information on the options that are supported.
`pathName window create index ?option value ...?`
This command creates a new window annotation, which will appear in the text at the position given by `index`. Any number of `option-value` pairs may be specified to configure the annotation. See **EMBEDDED WINDOWS** for information on the options that are supported. Returns an empty string.

`pathName window names`
Returns a list whose elements are the names of all windows currently embedded in `window`.

`pathName xview option args`
This command is used to query and change the horizontal position of the text in the widget's window. It can take any of the following forms:

`pathName xview`
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the portion of the document's horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the text is off-screen to the left, the middle 40% is visible in the window, and 40% of the text is off-screen to the right. The fractions refer only to the lines that are actually visible in the window: if the lines in the window are all very short, so that they are entirely visible, the returned fractions will be 0 and 1, even if there are other lines in the text that are much wider than the window. These are the same values passed to scrollbars via the `-xscrollcommand` option.

`pathName xview moveto fraction`
Adjusts the view in the window so that `fraction` of the horizontal span of the text is off-screen to the left. `Fraction` is a fraction between 0 and 1.

`pathName xview scroll number what`
This command shifts the view in the window left or right according to `number` and `what`. `What` must be **units**, **pages** or
pixels. If what is units or pages then number must be an integer, otherwise number may be specified in any of the forms acceptable to Tk_GetPixels, such as “2.0c” or “1i” (the result is rounded to the nearest integer value. If no units are given, pixels are assumed). If what is units, the view adjusts left or right by number average-width characters on the display; if it is pages then the view adjusts by number screenfuls; if it is pixels then the view adjusts by number pixels. If number is negative then characters farther to the left become visible; if it is positive then characters farther to the right become visible.

pathName yview ?args?

This command is used to query and change the vertical position of the text in the widget's window. It can take any of the following forms:

pathName yview

Returns a list containing two elements, both of which are real fractions between 0 and 1. The first element gives the position of the first visible pixel of the first character (or image, etc) in the top line in the window, relative to the text as a whole (0.5 means it is halfway through the text, for example). The second element gives the position of the first pixel just after the last visible one in the bottom line of the window, relative to the text as a whole. These are the same values passed to scrollbars via the -yscrollcommand option.

pathName yview moveto fraction

Adjusts the view in the window so that the pixel given by fraction appears at the top of the top line of the window. Fraction is a fraction between 0 and 1; 0 indicates the first pixel of the first character in the text, 0.33 indicates the pixel that is one-third the way through the text; and so on. Values close to 1 will indicate values close to the last pixel in the text (1 actually refers to one pixel beyond the last pixel), but in such cases the widget will never scroll beyond the last pixel, and so a value of 1 will effectively be rounded back to whatever fraction ensures the last pixel is at the bottom of the window,
and some other pixel is at the top.

\texttt{pathName yview scroll number what}

This command adjust the view in the window up or down according to \textit{number} and \textit{what}. \textit{What} must be \texttt{units}, \texttt{pages} or \texttt{pixels}. If \textit{what} is \texttt{units} or \texttt{pages} then \textit{number} must be an integer, otherwise number may be specified in any of the forms acceptable to \texttt{Tk\_GetPixels}, such as “2.0c” or “1i” (the result is rounded to the nearest integer value. If no units are given, pixels are assumed). If \textit{what} is \texttt{units}, the view adjusts up or down by \textit{number} lines on the display; if it is \texttt{pages} then the view adjusts by \textit{number} screenfuls; if it is \texttt{pixels} then the view adjusts by \textit{number} pixels. If \textit{number} is negative then earlier positions in the text become visible; if it is positive then later positions in the text become visible.

\texttt{pathName yview \(-\)pickplace\? index}

Changes the view in the widget's window to make \textit{index} visible. If the \texttt{-pickplace} option is not specified then \textit{index} will appear at the top of the window. If \texttt{-pickplace} is specified then the widget chooses where \textit{index} appears in the window:

[1] If \textit{index} is already visible somewhere in the window then the command does nothing.

[2] If \textit{index} is only a few lines off-screen above the window then it will be positioned at the top of the window.

[3] If \textit{index} is only a few lines off-screen below the window then it will be positioned at the bottom of the window.

[4] Otherwise, \textit{index} will be centered in the window.

The \texttt{-pickplace} option has been obsoleted by the \texttt{pathName}
see widget command (pathName see handles both x- and y-motion to make a location visible, whereas the -pickplace mode only handles motion in y).

***pathName yview number***
This command makes the first character on the line after the one given by number visible at the top of the window. Number must be an integer. This command used to be used for scrolling, but now it is obsolete.

---

**BINDINGS**

Tk automatically creates class bindings for texts that give them the following default behavior. In the descriptions below, “word” is dependent on the value of the tcl_wordchars variable. See tclvars(n).

[1] Clicking mouse button 1 positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget. Dragging with mouse button 1 strokes out a selection between the insertion cursor and the character under the mouse.

[2] Double-clicking with mouse button 1 selects the word under the mouse and positions the insertion cursor at the start of the word. Dragging after a double click will stroke out a selection consisting of whole words.

[3] Triple-clicking with mouse button 1 selects the line under the mouse and positions the insertion cursor at the start of the line. Dragging after a triple click will stroke out a selection consisting of whole lines.

[4] The ends of the selection can be adjusted by dragging with mouse button 1 while the Shift key is down; this will adjust the end of the
selection that was nearest to the mouse cursor when button 1 was pressed. If the button is double-clicked before dragging then the selection will be adjusted in units of whole words; if it is triple-clicked then the selection will be adjusted in units of whole lines.

[5] Clicking mouse button 1 with the Control key down will reposition the insertion cursor without affecting the selection.

[6] If any normal printing characters are typed, they are inserted at the point of the insertion cursor.

[7] The view in the widget can be adjusted by dragging with mouse button 2. If mouse button 2 is clicked without moving the mouse, the selection is copied into the text at the position of the mouse cursor. The Insert key also inserts the selection, but at the position of the insertion cursor.

[8] If the mouse is dragged out of the widget while button 1 is pressed, the entry will automatically scroll to make more text visible (if there is more text off-screen on the side where the mouse left the window).

[9] The Left and Right keys move the insertion cursor one character to the left or right; they also clear any selection in the text. If Left or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character. Control-Left and Control-Right move the insertion cursor by words, and Control-Shift-Left and Control-Shift-Right move the insertion cursor by words and also extend the selection. Control-b and Control-f behave the same as Left and Right, respectively. Meta-b and Meta-f behave the same as Control-Left and Control-Right, respectively.
The Up and Down keys move the insertion cursor one line up or down and clear any selection in the text. If Up or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character. Control-Up and Control-Down move the insertion cursor by paragraphs (groups of lines separated by blank lines), and Control-Shift-Up and Control-Shift-Down move the insertion cursor by paragraphs and also extend the selection. Control-p and Control-n behave the same as Up and Down, respectively.

The Next and Prior keys move the insertion cursor forward or backwards by one screenful and clear any selection in the text. If the Shift key is held down while Next or Prior is typed, then the selection is extended to include the new character.

Control-Next and Control-Prior scroll the view right or left by one page without moving the insertion cursor or affecting the selection.

Home and Control-a move the insertion cursor to the beginning of its display line and clear any selection in the widget. Shift-Home moves the insertion cursor to the beginning of the display line and also extends the selection to that point.

End and Control-e move the insertion cursor to the end of the display line and clear any selection in the widget. Shift-End moves the cursor to the end of the display line and extends the selection to that point.

Control-Home and Meta-< move the insertion cursor to the beginning of the text and clear any selection in the widget. Control-Shift-Home moves the insertion cursor to the beginning of the text and also extends the selection to that point.
Control-End and Meta- move the insertion cursor to the end of the text and clear any selection in the widget. Control-Shift-End moves the cursor to the end of the text and extends the selection to that point.

The Select key and Control-Space set the selection anchor to the position of the insertion cursor. They do not affect the current selection. Shift-Select and Control-Shift-Space adjust the selection to the current position of the insertion cursor, selecting from the anchor to the insertion cursor if there was not any selection previously.

Control-/ selects the entire contents of the widget.

Control-\ clears any selection in the widget.

The F16 key (labelled Copy on many Sun workstations) or Meta-w copies the selection in the widget to the clipboard, if there is a selection. This action is carried out by the command `tk_textCopy`.

The F20 key (labelled Cut on many Sun workstations) or Control-w copies the selection in the widget to the clipboard and deletes the selection. This action is carried out by the command `tk_textCut`. If there is no selection in the widget then these keys have no effect.

The F18 key (labelled Paste on many Sun workstations) or Control-y inserts the contents of the clipboard at the position of the insertion cursor. This action is carried out by the command `tk_textPaste`.

The Delete key deletes the selection, if there is one in the widget. If there is no selection, it deletes the character to the right of the insertion cursor.

[24]
Backspace and Control-h delete the selection, if there is one in the widget. If there is no selection, they delete the character to the left of the insertion cursor.

[25]
Control-d deletes the character to the right of the insertion cursor.

[26]
Meta-d deletes the word to the right of the insertion cursor.

[27]
Control-k deletes from the insertion cursor to the end of its line; if the insertion cursor is already at the end of a line, then Control-k deletes the newline character.

[28]
Control-o opens a new line by inserting a newline character in front of the insertion cursor without moving the insertion cursor.

[29]
Meta-backspace and Meta-Delete delete the word to the left of the insertion cursor.

[30]
Control-x deletes whatever is selected in the text widget after copying it to the clipboard.

[31]
Control-t reverses the order of the two characters to the right of the insertion cursor.

[32]
Control-z (and Control-underscore on UNIX when *tk_strictMotif* is true) undoes the last edit action if the *-undo* option is true. Does
nothing otherwise.

[33] Control-Z (or Control-y on Windows) reapplies the last undone edit action if the -undo option is true. Does nothing otherwise.

If the widget is disabled using the -state option, then its view can still be adjusted and text can still be selected, but no insertion cursor will be displayed and no text modifications will take place.

The behavior of texts can be changed by defining new bindings for individual widgets or by redefining the class bindings.

**KNOWN ISSUES**

**ISSUES CONCERNING CHARsms AND INDICES**

Before Tk 8.5, the widget used the string “chars” to refer to index positions (which included characters, embedded windows and embedded images). As of Tk 8.5 the text widget deals separately and correctly with “chars” and “indices”. For backwards compatibility, however, the index modifiers “+N chars” and “-N chars” continue to refer to indices. One must use any of the full forms “+N any chars” or “-N any chars” etc. to refer to actual character indices. This confusion may be fixed in a future release by making the widget correctly interpret “+N chars” as a synonym for “+N any chars”.

**PERFORMANCE ISSUES**

Text widgets should run efficiently under a variety of conditions. The text widget uses about 2-3 bytes of main memory for each byte of text, so texts containing a megabyte or more should be practical on most workstations. Text is represented internally with a modified B-tree structure that makes operations relatively efficient even with large texts. Tags are included in the B-tree structure in a way that allows tags to span large ranges or have many disjoint smaller ranges without loss of efficiency. Marks are also implemented in a way that allows large numbers of marks. In most cases it is fine to have large numbers of
unique tags, or a tag that has many distinct ranges.

One performance problem can arise if you have hundreds or thousands of different tags that all have the following characteristics: the first and last ranges of each tag are near the beginning and end of the text, respectively, or a single tag range covers most of the text widget. The cost of adding and deleting tags like this is proportional to the number of other tags with the same properties. In contrast, there is no problem with having thousands of distinct tags if their overall ranges are localized and spread uniformly throughout the text.

Very long text lines can be expensive, especially if they have many marks and tags within them.

The display line with the insert cursor is redrawn each time the cursor blinks, which causes a steady stream of graphics traffic. Set the insertOffTime attribute to 0 to avoid this.

**KNOWN BUGS**

The `pathName search -regexp` sub-command attempts to perform sophisticated regexp matching across multiple lines in an efficient fashion (since Tk 8.5), examining each line individually, and then in small groups of lines, whether searching forwards or backwards. Under certain conditions the search result might differ from that obtained by applying the same regexp to the entire text from the widget in one go. For example, when searching with a greedy regexp, the widget will continue to attempt to add extra lines to the match as long as one of two conditions are true: either Tcl's regexp library returns a code to indicate a longer match is possible (but there are known bugs in Tcl which mean this code is not always correctly returned); or if each extra line added results in at least a partial match with the pattern. This means in the case where the first extra line added results in no match and Tcl's regexp system returns the incorrect code and adding a second extra line would actually match, the text widget will return the wrong result. In practice this is a rare problem, but it can occur, for example:
will not find a match when one exists of 19 characters starting from the first “b”.

Whenever one possible match is fully enclosed in another, the search command will attempt to ensure only the larger match is returned. When performing backwards regexp searches it is possible that Tcl will not always achieve this, in the case where a match is preceded by one or more short, non-overlapping matches, all of which are preceded by a large match which actually encompasses all of them. The search algorithm used by the widget does not look back arbitrarily far for a possible match which might cover large portions of the widget. For example:

matches at “5.0” when a true greedy match would match at “1.0”. Similarly if we add -all to this case, it matches at all of “5.0”, “4.0”, “3.0” and “1.0”, when really it should only match at “1.0” since that match encloses all the others.

SEE ALSO

entry, scrollbar

KEYWORDS

text, widget, tkvars
NAME
ttk::panedwindow - Multi-pane container window

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS
  -class
  -cursor, cursor, Cursor
  -style
  -takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS
  -orient, orient, Orient
  -width, width, Width
  -height, height, Height

PANE OPTIONS
  -weight, weight, Weight

WIDGET COMMAND

  pathname add subwindow options...
  pathname forget pane
  pathname identify x y
  pathname insert pos subwindow options...
  pathname pane pane -option ?value ?-option value...
  pathname sashpos index ?newpos?

SEE ALSO

NAME
ttk::panedwindow - Multi-pane container window

SYNOPSIS

ttk::panedwindow pathName ?options?
  pathName add window ?options...?
A `ttk::panedwindow` widget displays a number of subwindows, stacked either vertically or horizontally. The user may adjust the relative sizes of the subwindows by dragging the sash between panes.

### STANDARD OPTIONS

- `-class`
- `-cursor, cursor, Cursor`
- `-style`
- `-takefocus, takeFocus, TakeFocus`

### WIDGET-SPECIFIC OPTIONS

**Command-Line Name:** `-orient`
**Database Name:** `orient`
**Database Class:** `Orient`
   - Specifies the orientation of the window. If **vertical**, subpanes are stacked top-to-bottom; if **horizontal**, subpanes are stacked left-to-right.

**Command-Line Name:** `-width`
**Database Name:** `width`
**Database Class:** `Width`
   - If present and greater than zero, specifies the desired width of the widget in pixels. Otherwise, the requested width is determined by the width of the managed windows.

**Command-Line Name:** `-height`
**Database Name:** `height`
**Database Class:** `Height`
   - If present and greater than zero, specifies the desired height of the widget in pixels. Otherwise, the requested height is determined by the height of the managed windows.
## PANE OPTIONS

The following options may be specified for each pane:

**Command-Line Name:** `-weight`
**Database Name:** `weight`
**Database Class:** `Weight`

An integer specifying the relative stretchability of the pane. When the paned window is resized, the extra space is added or subtracted to each pane proportionally to its `-weight`.

## WIDGET COMMAND

Supports the standard `configure`, `cget`, `state`, and `instate` commands; see `ttk::widget(n)` for details. Additional commands:

`pathname add subwindow options...`
Add a new pane to the window. `subwindow` must be a direct child of the paned window `pathname`. See `PANE OPTIONS` for the list of available options.

`pathname forget pane`
Removes the specified subpane from the widget. `pane` is either an integer index or the name of a managed subwindow.

`pathname identify x y`
Returns the index of the sash at point `x,y`, or the empty string if `x,y` is not over a sash.

`pathname insert pos subwindow options...`
Inserts a pane at the specified position. `pos` is either the string `end`, an integer index, or the name of a managed subwindow. If `subwindow` is already managed by the paned window, moves it to the specified position. See `PANE OPTIONS` for the list of available options.

`pathname pane pane -option ?value ?-option value...`
Query or modify the options of the specified `pane`, where `pane` is
either an integer index or the name of a managed subwindow. If no -option is specified, returns a dictionary of the pane option values. If one -option is specified, returns the value of that option. Otherwise, sets the -options to the corresponding values.

pathname sashpos index ?newpos?
If newpos is specified, sets the position of sash number index. May adjust the positions of adjacent sashes to ensure that positions are monotonically increasing. Sash positions are further constrained to be between 0 and the total size of the widget. Returns the new position of sash number index.

SEE ALSO

ttk::widget, ttk::notebook, panedwindow

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2005 Joe English
NAME
bind - Arrange for X events to invoke Tcl scripts

SYNOPSIS

INTRODUCTION

EVENT PATTERNS
MODIFIERS

EVENT TYPES
Activate, Deactivate
MouseWheel
KeyPress, KeyRelease
ButtonPress, ButtonRelease, Motion
Configure
Map, Unmap
Visibility
Expose
Destroy
FocusIn, FocusOut
Enter, Leave
Property
Colormap
MapRequest, CirculateRequest, ResizeRequest,
ConfigureRequest, Create
Gravity, Reparent, Circulate

EVENT DETAILS
BINDING SCRIPTS AND SUBSTITUTIONS
%%
%#
%a
%b
%c
%d
MULTIPLE MATCHES
(a)
(b)
(c)
(d)
(e)
MULTI-EVENT SEQUENCES AND IGNORED EVENTS
ERRORS
EXAMPLES
SEE ALSO
KEYWORDS
bind - Arrange for X events to invoke Tcl scripts

SYNOPSIS

bind tag ?sequence? ?+??script?

INTRODUCTION

The bind command associates Tcl scripts with X events. If all three arguments are specified, bind will arrange for script (a Tcl script) to be evaluated whenever the event(s) given by sequence occur in the window(s) identified by tag. If script is prefixed with a “+”, then it is appended to any existing binding for sequence; otherwise script replaces any existing binding. If script is an empty string then the current binding for sequence is destroyed, leaving sequence unbound. In all of the cases where a script argument is provided, bind returns an empty string.

If sequence is specified without a script, then the script currently bound to sequence is returned, or an empty string is returned if there is no binding for sequence. If neither sequence nor script is specified, then the return value is a list whose elements are all the sequences for which there exist bindings for tag.

The tag argument determines which window(s) the binding applies to. If tag begins with a dot, as in .a.b.c, then it must be the path name for a window; otherwise it may be an arbitrary string. Each window has an associated list of tags, and a binding applies to a particular window if its tag is among those specified for the window. Although the bindtags command may be used to assign an arbitrary set of binding tags to a window, the default binding tags provide the following behavior:

- If a tag is the name of an internal window the binding applies to that window.
- If the tag is the name of a toplevel window the binding applies to the toplevel window and all its internal windows.
• If the tag is the name of a class of widgets, such as Button, the binding applies to all widgets in that class;

• If tag has the value all, the binding applies to all windows in the application.

EVENT PATTERNS

The sequence argument specifies a sequence of one or more event patterns, with optional white space between the patterns. Each event pattern may take one of three forms. In the simplest case it is a single printing ASCII character, such as a or [. The character may not be a space character or the character <. This form of pattern matches a KeyPress event for the particular character. The second form of pattern is longer but more general. It has the following syntax:

<modifier-modifier-type-detail>

The entire event pattern is surrounded by angle brackets. Inside the angle brackets are zero or more modifiers, an event type, and an extra piece of information (detail) identifying a particular button or keysym. Any of the fields may be omitted, as long as at least one of type and detail is present. The fields must be separated by white space or dashes.

The third form of pattern is used to specify a user-defined, named virtual event. It has the following syntax:

<<name>>

The entire virtual event pattern is surrounded by double angle brackets. Inside the angle brackets is the user-defined name of the virtual event. Modifiers, such as Shift or Control, may not be combined with a virtual event to modify it. Bindings on a virtual event may be created before the virtual event is defined, and if the definition of a virtual event changes
dynamically, all windows bound to that virtual event will respond immediately to the new definition.

Some widgets (e.g. menu and text) issue virtual events when their internal state is updated in some ways. Please see the manual page for each widget for details.

**MODIFIERS**

Modifiers consist of any of the following values:

<table>
<thead>
<tr>
<th>Modifier</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>Mod1, M1, Command</td>
</tr>
<tr>
<td>Alt</td>
<td>Mod2, M2, Option</td>
</tr>
<tr>
<td>Shift</td>
<td>Mod3, M3</td>
</tr>
<tr>
<td>Lock</td>
<td>Mod4, M4</td>
</tr>
<tr>
<td>Extended</td>
<td>Mod5, M5</td>
</tr>
<tr>
<td>Button1, B1</td>
<td>Meta, M</td>
</tr>
<tr>
<td>Button2, B2</td>
<td>Double</td>
</tr>
<tr>
<td>Button3, B3</td>
<td>Triple</td>
</tr>
<tr>
<td>Button4, B4</td>
<td>Quadruple</td>
</tr>
</tbody>
</table>
Button5, B5

Where more than one value is listed, separated by commas, the values are equivalent. Most of the modifiers have the obvious X meanings. For example, Button1 requires that button 1 be depressed when the event occurs. For a binding to match a given event, the modifiers in the event must include all of those specified in the event pattern. An event may also contain additional modifiers not specified in the binding. For example, if button 1 is pressed while the shift and control keys are down, the pattern <Control-Button-1> will match the event, but <Mod1-Button-1> will not. If no modifiers are specified, then any combination of modifiers may be present in the event.

Meta and M refer to whichever of the M1 through M5 modifiers is associated with the Meta key(s) on the keyboard (keysym Meta_R and Meta_L). If there are no Meta keys, or if they are not associated with any modifiers, then Meta and M will not match any events. Similarly, the Alt modifier refers to whichever modifier is associated with the alt key(s) on the keyboard (keysym Alt_L and Alt_R).

The Double, Triple and Quadruple modifiers are a convenience for specifying double mouse clicks and other repeated events. They cause a particular event pattern to be repeated 2, 3 or 4 times, and also place a time and space requirement on the sequence: for a sequence of events to match a Double, Triple or Quadruple pattern, all of the events must occur close together in time and without substantial mouse motion in between. For example, <Double-Button-1> is equivalent to <Button-1><Button-1> with the extra time and space requirement.

The Command and Option modifiers are equivalents of Mod1 resp. Mod2, they correspond to Macintosh-specific modifier keys.

The Extended modifier is, at present, specific to Windows. It appears on events that are associated with the keys on the “extended keyboard”. On a US keyboard, the extended keys include the Alt and Control keys at the right of the keyboard, the cursor keys in the cluster to the left of the numeric pad, the NumLock key, the Break key, the
PrintScreen key, and the / and Enter keys in the numeric keypad.

**EVENT TYPES**

The *type* field may be any of the standard X event types, with a few extra abbreviations. The *type* field will also accept a couple non-standard X event types that were added to better support the Macintosh and Windows platforms. Below is a list of all the valid types; where two names appear together, they are synonyms.

<table>
<thead>
<tr>
<th>Activate</th>
<th>Destroy</th>
<th>Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>ButtonPress, <strong>Button</strong></td>
<td>Enter</td>
<td>MapRequest</td>
</tr>
<tr>
<td>ButtonRelease</td>
<td>Expose</td>
<td>Motion</td>
</tr>
<tr>
<td>Circulate</td>
<td>FocusIn</td>
<td>MouseWheel</td>
</tr>
<tr>
<td>CirculateRequest</td>
<td>FocusOut</td>
<td>Property</td>
</tr>
<tr>
<td>Colormap</td>
<td>Gravity</td>
<td>Reparent</td>
</tr>
<tr>
<td>Configure</td>
<td>KeyPress, Key</td>
<td>ResizeRequest</td>
</tr>
<tr>
<td>ConfigureRequest</td>
<td>KeyRelease</td>
<td>Unmap</td>
</tr>
<tr>
<td>Create</td>
<td>Leave</td>
<td>Visibility</td>
</tr>
<tr>
<td>Deactivate</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Most of the above events have the same fields and behaviors as events in the X Windowing system. You can find more detailed descriptions of these events in any X window programming book. A couple of the events are extensions to the X event system to support features unique to the Macintosh and Windows platforms. We provide a little more detail on these events here. These include:

**Activate, Deactivate**

These two events are sent to every sub-window of a toplevel when they change state. In addition to the focus Window, the Macintosh platform and Windows platforms have a notion of an active window (which often has but is not required to have the focus). On the Macintosh, widgets in the active window have a different appearance than widgets in deactivate windows. The **Activate** event is sent to all the sub-windows in a toplevel when it changes from being deactivate to active. Likewise, the **Deactive** event is sent when the window's state changes from active to deactivate. There are no useful percent substitutions you would make when binding to these events.

**MouseWheel**

Many contemporary mice support a mouse wheel, which is used for scrolling documents without using the scrollbars. By rolling the wheel, the system will generate **MouseWheel** events that the application can use to scroll. Like **Key** events the event is always routed to the window that currently has focus. When the event is received you can use the `%d` substitution to get the `delta` field for the event, which is a integer value describing how the mouse wheel has moved. The smallest value for which the system will report is defined by the OS. On Windows 95 & 98 machines this value is at least 120 before it is reported. However, higher resolution devices may be available in the future. The sign of the value determines which direction your widget should scroll. Positive values should scroll up and negative values should scroll down.

**KeyPress, KeyRelease**
The KeyPress and KeyRelease events are generated whenever a key is pressed or released. KeyPress and KeyRelease events are sent to the window which currently has the keyboard focus.

**ButtonPress, ButtonRelease, Motion**

The ButtonPress and ButtonRelease events are generated when the user presses or releases a mouse button. Motion events are generated whenever the pointer is moved. ButtonPress, ButtonRelease, and Motion events are normally sent to the window containing the pointer.

When a mouse button is pressed, the window containing the pointer automatically obtains a temporary pointer grab. Subsequent ButtonPress, ButtonRelease, and Motion events will be sent to that window, regardless of which window contains the pointer, until all buttons have been released.

**Configure**

A Configure event is sent to a window whenever its size, position, or border width changes, and sometimes when it has changed position in the stacking order.

**Map, Unmap**

The Map and Unmap events are generated whenever the mapping state of a window changes.

Windows are created in the unmapped state. Top-level windows become mapped when they transition to the normal state, and are unmapped in the withdrawn and iconic states. Other windows become mapped when they are placed under control of a geometry manager (for example pack or grid).

A window is viewable only if it and all of its ancestors are mapped. Note that geometry managers typically do not map their children until they have been mapped themselves, and unmap all children when they become unmapped; hence in Tk Map and Unmap events indicate whether or not a window is viewable.
Visibility
A window is said to be obscured when another window above it in the stacking order fully or partially overlaps it. Visibility events are generated whenever a window’s obscurity state changes; the state field (%s) specifies the new state.

Expose
An Expose event is generated whenever all or part of a window should be redrawn (for example, when a window is first mapped or if it becomes unobscured). It is normally not necessary for client applications to handle Expose events, since Tk handles them internally.

Destroy
A Destroy event is delivered to a window when it is destroyed.

When the Destroy event is delivered to a widget, it is in a “half-dead” state: the widget still exists, but most operations on it will fail.

FocusIn, FocusOut
The FocusIn and FocusOut events are generated whenever the keyboard focus changes. A FocusOut event is sent to the old focus window, and a FocusIn event is sent to the new one.

In addition, if the old and new focus windows do not share a common parent, “virtual crossing” focus events are sent to the intermediate windows in the hierarchy. Thus a FocusIn event indicates that the target window or one of its descendants has acquired the focus, and a FocusOut event indicates that the focus has been changed to a window outside the target window's hierarchy.

The keyboard focus may be changed explicitly by a call to focus, or implicitly by the window manager.

Enter, Leave
An Enter event is sent to a window when the pointer enters that window, and a Leave event is sent when the pointer leaves it.
If there is a pointer grab in effect, **Enter** and **Leave** events are only delivered to the window owning the grab.

In addition, when the pointer moves between two windows, **Enter** and **Leave** “virtual crossing” events are sent to intermediate windows in the hierarchy in the same manner as for **FocusIn** and **FocusOut** events.

**Property**

A **Property** event is sent to a window whenever an X property belonging to that window is changed or deleted. **Property** events are not normally delivered to Tk applications as they are handled by the Tk core.

**Colormap**

A **Colormap** event is generated whenever the colormap associated with a window has been changed, installed, or uninstalled.

Widgets may be assigned a private colormap by specifying a `-colormap` option; the window manager is responsible for installing and uninstalling colormaps as necessary.

Note that Tk provides no useful details for this event type.

**MapRequest, CirculateRequest, ResizeRequest, ConfigureRequest, Create**

These events are not normally delivered to Tk applications. They are included for completeness, to make it possible to write X11 window managers in Tk. (These events are only delivered when a client has selected **SubstructureRedirectMask** on a window; the Tk core does not use this mask.)

**Gravity, Reparent, Circulate**

The events **Gravity** and **Reparent** are not normally delivered to Tk applications. They are included for completeness.

A **Circulate** event indicates that the window has moved to the top or to the bottom of the stacking order as a result of an
X CirculateSubwindows protocol request. Note that the stacking order may be changed for other reasons which do not generate a Circulate event, and that Tk does not use X CirculateSubwindows() internally. This event type is included only for completeness; there is no reliable way to track changes to a window’s position in the stacking order.

**EVENT DETAILS**

The last part of a long event specification is *detail*. In the case of a ButtonPress or ButtonRelease event, it is the number of a button (1-5). If a button number is given, then only an event on that particular button will match; if no button number is given, then an event on any button will match. Note: giving a specific button number is different than specifying a button modifier; in the first case, it refers to a button being pressed or released, while in the second it refers to some other button that is already depressed when the matching event occurs. If a button number is given then *type* may be omitted: if will default to ButtonPress. For example, the specifier `<1>` is equivalent to `<ButtonPress-1>`.

If the event type is KeyPress or KeyRelease, then *detail* may be specified in the form of an X keysym. Keysyms are textual specifications for particular keys on the keyboard; they include all the alphanumeric ASCII characters (e.g. “a” is the keysym for the ASCII character “a”), plus descriptions for non-alphanumeric characters (“comma”is the keysym for the comma character), plus descriptions for all the non-ASCII keys on the keyboard (e.g. “Shift_L” is the keysym for the left shift key, and “F1” is the keysym for the F1 function key, if it exists). The complete list of keysyms is not presented here; it is available in other X documentation and may vary from system to system. If necessary, you can use the %K notation described below to print out the keysym name for a particular key. If a keysym *detail* is given, then the *type* field may be omitted; it will default to KeyPress. For example, `<Control-comma>` is equivalent to `<Control-KeyPress-comma>`.
The *script* argument to *bind* is a Tcl script, which will be executed whenever the given event sequence occurs. *Command* will be executed in the same interpreter that the *bind* command was executed in, and it will run at global level (only global variables will be accessible). If *script* contains any % characters, then the script will not be executed directly. Instead, a new script will be generated by replacing each %, and the character following it, with information from the current event. The replacement depends on the character following the %, as defined in the list below. Unless otherwise indicated, the replacement string is the decimal value of the given field from the current event. Some of the substitutions are only valid for certain types of events; if they are used for other types of events the value substituted is undefined.

%%
Replaced with a single percent.

%#
The number of the last client request processed by the server (the *serial* field from the event). Valid for all event types.

%a
The *above* field from the event, formatted as a hexadecimal number. Valid only for *Configure* events. Indicates the sibling window immediately below the receiving window in the stacking order, or 0 if the receiving window is at the bottom.

%b
The number of the button that was pressed or released. Valid only for *ButtonPress* and *ButtonRelease* events.

%c
The *count* field from the event. Valid only for *Expose* events. Indicates that there are *count* pending *Expose* events which have not yet been delivered to the window.

%d
The *detail* or *user_data* field from the event. The `%%d` is replaced by a string identifying the detail. For *Enter*, *Leave*, *FocusIn*, and *FocusOut* events, the string will be one of the following:

- NotifyAncestor
- NotifyNonlinearVirtual
- NotifyDetailNone
- NotifyPointer
- NotifyInferior
- NotifyPointerRoot
- NotifyNonlinear
- NotifyVirtual

For *ConfigureRequest* events, the string will be one of:

- Above
- Opposite
- Below
- None
- BottomIf
- TopIf

For virtual events, the string will be whatever value is stored in the *user_data* field when the event was created (typically with `event generate`), or the empty string if the field is NULL. Virtual events corresponding to key sequence presses (see *event add* for details) set the *user_data* to NULL. For events other than these, the substituted string is undefined.

`%%f`

The *focus* field from the event (0 or 1). Valid only for *Enter* and *Leave* events. 1 if the receiving window is the focus window or a descendant of the focus window, 0 otherwise.

`%%h`
The *height* field from the event. Valid for the `Configure`, `ConfigureRequest`, `Create`, `ResizeRequest`, and `Expose` events. Indicates the new or requested height of the window.

%i
The *window* field from the event, represented as a hexadecimal integer. Valid for all event types.

%k
The *keycode* field from the event. Valid only for `KeyPress` and `KeyRelease` events.

%m
The *mode* field from the event. The substituted string is one of `NotifyNormal`, `NotifyGrab`, `NotifyUngrab`, or `NotifyWhileGrabbed`. Valid only for `Enter`, `FocusIn`, `FocusOut`, and `Leave` events.

%o
The *override_redirect* field from the event. Valid only for `Map`, `Reparent`, and `Configure` events.

%p
The *place* field from the event, substituted as one of the strings `PlaceOnTop` or `PlaceOnBottom`. Valid only for `Circulate` and `CirculateRequest` events.

%s
The *state* field from the event. For `ButtonPress`, `ButtonRelease`, `Enter`, `KeyPress`, `KeyRelease`, `Leave`, and `Motion` events, a decimal string is substituted. For `Visibility`, one of the strings `VisibilityUnobscured`, `VisibilityPartiallyObscured`, and `VisibilityFullyObscured` is substituted. For `Property` events, substituted with either the string `NewValue` (indicating that the property has been created or modified) or `Delete` (indicating that the property has been removed).

%t
The *time* field from the event. This is the X server timestamp (typically the time since the last server reset) in milliseconds, when the event occurred. Valid for most events.

**%w**

The *width* field from the event. Indicates the new or requested width of the window. Valid only for *Configure, ConfigureRequest, Create, ResizeRequest*, and *Expose* events.

**%x, %y**

The *x* and *y* fields from the event. For *ButtonPress, ButtonRelease, Motion, KeyPress, KeyRelease*, and *MouseWheel* events, %x and %y indicate the position of the mouse pointer relative to the receiving window. For *Enter and Leave* events, the position where the mouse pointer crossed the window, relative to the receiving window. For *Configure and Create* requests, the *x* and *y* coordinates of the window relative to its parent window.

**%A**

Substitutes the UNICODE character corresponding to the event, or the empty string if the event does not correspond to a UNICODE character (e.g. the shift key was pressed). *XmbLookupString* (or *XLookupString* when input method support is turned off) does all the work of translating from the event to a UNICODE character. Valid only for *KeyPress and KeyRelease* events.

**%B**

The *border_width* field from the event. Valid only for *Configure, ConfigureRequest*, and *Create* events.

**%D**

This reports the *delta* value of a *MouseWheel* event. The *delta* value represents the rotation units the mouse wheel has been moved. On Windows 95 & 98 systems the smallest value for the delta is 120. Future systems may support higher resolution values for the delta. The sign of the value represents the direction the mouse wheel was scrolled.
%E
The send_event field from the event. Valid for all event types. 0 indicates that this is a “normal” event, 1 indicates that it is a “synthetic” event generated by SendEvent.

%K
The keysym corresponding to the event, substituted as a textual string. Valid only for KeyPress and KeyRelease events.

%N
The keysym corresponding to the event, substituted as a decimal number. Valid only for KeyPress and KeyRelease events.

%P
The name of the property being updated or deleted (which may be converted to an XAtom using \texttt{winfo atom}.) Valid only for Property events.

%R
The root window identifier from the event. Valid only for events containing a root field.

%S
The subwindow window identifier from the event, formatted as a hexadecimal number. Valid only for events containing a subwindow field.

%T
The type field from the event. Valid for all event types.

%W
The path name of the window to which the event was reported (the window field from the event). Valid for all event types.

%X, %Y
The x_root and y_root fields from the event. If a virtual-root window manager is being used then the substituted values are the corresponding x-coordinate and y-coordinate in the virtual root.
Valid only for **ButtonPress**, **ButtonRelease**, **KeyPress**, **KeyRelease**, and **Motion** events. Same meaning as `%x` and `%y`, except relative to the (virtual) root window.

The replacement string for a `%`-replacement is formatted as a proper Tcl list element. This means that spaces or special characters such as $ and { may be preceded by backslashes. This guarantees that the string will be passed through the Tcl parser when the binding script is evaluated. Most replacements are numbers or well-defined strings such as **Above**; for these replacements no special formatting is ever necessary. The most common case where reformatting occurs is for the `%A` substitution. For example, if *script* is

```tcl
insert %A
```

and the character typed is an open square bracket, then the script actually executed will be

```tcl
insert \[
```

This will cause the *insert* to receive the original replacement string (open square bracket) as its first argument. If the extra backslash had not been added, Tcl would not have been able to parse the script correctly.

### MULTIPLE MATCHES

It is possible for several bindings to match a given X event. If the bindings are associated with different *tag*’s, then each of the bindings will be executed, in order. By default, a binding for the widget will be executed first, followed by a class binding, a binding for its toplevel, and an **all** binding. The *bindtags* command may be used to change this order for a particular window or to associate additional binding tags with the window.
The **continue** and **break** commands may be used inside a binding script to control the processing of matching scripts. If **continue** is invoked, then the current binding script is terminated but Tk will continue processing binding scripts associated with other tag's. If the **break** command is invoked within a binding script, then that script terminates and no other scripts will be invoked for the event.

If more than one binding matches a particular event and they have the same *tag*, then the most specific binding is chosen and its script is evaluated. The following tests are applied, in order, to determine which of several matching sequences is more specific:

(a) an event pattern that specifies a specific button or key is more specific than one that does not;

(b) a longer sequence (in terms of number of events matched) is more specific than a shorter sequence;

(c) if the modifiers specified in one pattern are a subset of the modifiers in another pattern, then the pattern with more modifiers is more specific.

(d) a virtual event whose physical pattern matches the sequence is less specific than the same physical pattern that is not associated with a virtual event.

(e) given a sequence that matches two or more virtual events, one of the virtual events will be chosen, but the order is undefined.

If the matching sequences contain more than one event, then tests (c)-(e) are applied in order from the most recent event to the least recent event in the sequences. If these tests fail to determine a winner, then the most recently registered sequence is the winner.
If there are two (or more) virtual events that are both triggered by the same sequence, and both of those virtual events are bound to the same window tag, then only one of the virtual events will be triggered, and it will be picked at random:

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>event add &lt;&lt;Paste&gt;&gt; &lt;Control-y&gt;</code></td>
<td>Paste event triggered by Control-y combination</td>
</tr>
<tr>
<td><code>event add &lt;&lt;Paste&gt;&gt; &lt;Button-2&gt;</code></td>
<td>Paste event triggered by button 2</td>
</tr>
<tr>
<td><code>event add &lt;&lt;Scroll&gt;&gt; &lt;Button-2&gt;</code></td>
<td>Scroll event triggered by button 2</td>
</tr>
<tr>
<td><code>bind Entry &lt;&lt;Paste&gt;&gt; {puts Paste}</code></td>
<td>Bind Paste event to an entry field</td>
</tr>
<tr>
<td><code>bind Entry &lt;&lt;Scroll&gt;&gt; {puts Scroll}</code></td>
<td>Bind Scroll event to an entry field</td>
</tr>
</tbody>
</table>

If the user types Control-y, the `<<paste>>` binding will be invoked, but if the user presses button 2 then one of either the `<<paste>>` or the `<<scroll>>` bindings will be invoked, but exactly which one gets invoked is undefined.

If an X event does not match any of the existing bindings, then the event is ignored. An unbound event is not considered to be an error.

**MULTI-EVENT SEQUENCES AND IGNORED EVENTS**

When a sequence specified in a `bind` command contains more than one event pattern, then its script is executed whenever the recent events (leading up to and including the current event) match the given sequence. This means, for example, that if button 1 is clicked repeatedly the sequence `<<Double-ButtonPress-1>>` will match each button press but the first. If extraneous events that would prevent a match occur in the middle of an event sequence then the extraneous events are ignored unless they are `KeyPress` or `ButtonPress` events. For example, `<<Double-ButtonPress-1>>` will match a sequence of presses of button 1, even though there will be `ButtonRelease` events (and possibly `Motion` events) between the `ButtonPress` events. Furthermore, a `KeyPress` event may be preceded by any number of other `KeyPress` events for modifier keys without the modifier keys preventing a match. For example, the event sequence `aB` will match a press of the `a` key, a release of the `a` key, a press of the `Shift` key, and a
press of the b key: the press of Shift is ignored because it is a modifier key. Finally, if several Motion events occur in a row, only the last one is used for purposes of matching binding sequences.

**ERRORS**

If an error occurs in executing the script for a binding then the bgerror mechanism is used to report the error. The bgerror command will be executed at global level (outside the context of any Tcl procedure).

**EXAMPLES**

Arrange for a string describing the motion of the mouse to be printed out when the mouse is double-clicked:

```tcl
bind . <Double-1> {
    puts "hi from (%x,%y)"
}
```

A little GUI that displays what the keysym name of the last key pressed is:

```tcl
set keysym "Press any key"
pack [label .1 -textvariable keysym -padx 2m -pady 1
bind . <Key> {
    set keysym "You pressed %K"
}
```

**SEE ALSO**

bgerror, bindtags, event, focus, grab, keysyms

**KEYWORDS**
binding, event
NAME
frame - Create and manipulate frame widgets

SYNOPSIS

STANDARD OPTIONS
-`-borderwidth` or `-bd`, `borderWidth`, `BorderWidth`  
-`-cursor`, `cursor`, `Cursor`  
-`-highlightbackground`, `highlightBackground`, `HighlightBackground`  
-`-highlightcolor`, `highlightColor`, `HighlightColor`  
-`-highlightthickness`, `highlightThickness`, `HighlightThickness`  
-`-padx`, `padX`, `Pad`  
-`-pady`, `padY`, `Pad`  
-`-relief`, `relief`, `Relief`  
-`-takefocus`, `takeFocus`, `TakeFocus`  

WIDGET-SPECIFIC OPTIONS
-`-background`, `background`, `Background`  
-`-class`, `class`, `Class`  
-`-colormap`, `colormap`, `Colormap`  
-`-container`, `container`, `Container`  
-`-height`, `height`, `Height`  
-`-visual`, `visual`, `Visual`  
-`-width`, `width`, `Width`  

DESCRIPTION

WIDGET COMMAND
```
pathName cget option
```
```
pathName configure ?option? ?value option value ...?
```

BINDINGS

SEE ALSO

KEYWORDS

NAME
frame - Create and manipulate frame widgets

**SYNOPSIS**

`frame pathName ?options?`

**STANDARD OPTIONS**

- `-borderwidth or -bd, borderWidth, BorderWidth`
- `-cursor, cursor, Cursor`
- `-highlightbackground, highlightBackground, HighlightBackground`
- `-highlightcolor, highlightColor, HighlightColor`
- `-highlightthickness, highlightThickness, HighlightThickness`
- `-padx, padX, Pad`
- `-pady, padY, Pad`
- `-relief, relief, Relief`
- `-takefocus, takeFocus, TakeFocus`

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: `-background`
Database Name: `background`
Database Class: `Background`
This option is the same as the standard `background` option except that its value may also be specified as an empty string. In this case, the widget will display no background or border, and no colors will be consumed from its colormap for its background and border.

Command-Line Name: `-class`
Database Name: `class`
Database Class: `Class`
Specifies a class for the window. This class will be used when querying the option database for the window's other options, and it will also be used later for other purposes such as bindings. The `class` option may not be changed with the `configure` widget command.

Command-Line Name: `-colormap`
Database Name: **colormap**
Database Class: **Colormap**
Specifies a colormap to use for the window. The value may be either new, in which case a new colormap is created for the window and its children, or the name of another window (which must be on the same screen and have the same visual as `pathName`), in which case the new window will use the colormap from the specified window. If the **colormap** option is not specified, the new window uses the same colormap as its parent. This option may not be changed with the `configure` widget command.

Command-Line Name: **-container**
Database Name: **container**
Database Class: **Container**
The value must be a boolean. If true, it means that this window will be used as a container in which some other application will be embedded (for example, a Tk toplevel can be embedded using the **-use** option). The window will support the appropriate window manager protocols for things like geometry requests. The window should not have any children of its own in this application. This option may not be changed with the `configure` widget command.

Command-Line Name: **-height**
Database Name: **height**
Database Class: **Height**
Specifies the desired height for the window in any of the forms acceptable to `Tk_GetPixels`. If this option is less than or equal to zero then the window will not request any size at all. Note that this sets the total height of the frame, any **-borderwidth** or similar is not added. Normally **-height** should not be used if a propagating geometry manager, such as `grid` or `pack`, is used within the frame since the geometry manager will override the height of the frame.

Command-Line Name: **-visual**
Database Name: **visual**
Database Class: **Visual**
Specifies visual information for the new window in any of the forms accepted by `Tk_GetVisual`. If this option is not specified, the new
window will use the same visual as its parent. The **visual** option may not be modified with the **configure** widget command.

Command-Line Name: **-width**  
Database Name: **width**  
Database Class: **Width**  

Specifies the desired width for the window in any of the forms acceptable to [Tk GetPixels](https://www.tcl-lang.org/man/tcl8.6/ulman/tcl8.6.tkhtml/man/tcl8.6/TkCmd/tk_getpixels.htm). If this option is less than or equal to zero then the window will not request any size at all. Note that this sets the total width of the frame, any **-borderwidth** or similar is not added. Normally **-width** should not be used if a propagating geometry manager, such as [grid](https://www.tcl-lang.org/man/tcl8.6/ulman/tcl8.6.tkhtml/man/tcl8.6/TkCmd/grid.htm) or [pack](https://www.tcl-lang.org/man/tcl8.6/ulman/tcl8.6.tkhtml/man/tcl8.6/TkCmd/pack.htm), is used within the frame since the geometry manager will override the width of the frame.

### DESCRIPTION

The **frame** command creates a new window (given by the **pathName** argument) and makes it into a frame widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the frame such as its background color and relief. The **frame** command returns the path name of the new window.

A frame is a simple widget. Its primary purpose is to act as a spacer or container for complex window layouts. The only features of a frame are its background color and an optional 3-D border to make the frame appear raised or sunken.

### WIDGET COMMAND

The **frame** command creates a new Tcl command whose name is the same as the path name of the frame's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```
PathName is the name of the command, which is the same as the frame widget's path name. Option and the args determine the exact behavior of the command. The following commands are possible for frame widgets:

**pathName cget option**

Returns the current value of the configuration option given by option. Option may have any of the values accepted by the frame command.

**pathName configure ?option? ?value option value ...?**

Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option-value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the frame command.

**BINDINGS**

When a new frame is created, it has no default event bindings: frames are not intended to be interactive.

**SEE ALSO**

labelframe, toplevel, ttk::frame

**KEYWORDS**

frame, widget

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
pack - Geometry manager that packs around edges of cavity

SYNOPSIS
DESCRIPTION
pack slave ?slave ...? ?options?
pack configure slave ?slave ...? ?options?
   -after other
   -anchor anchor
   -before other
   -expand boolean
   -fill style
       none
       x
       y
       both
   -in other
   -ipadx amount
   -ipady amount
   -padx amount
   -pady amount
   -side side
pack forget slave ?slave ...?
pack info slave
pack propagate master ?boolean?
pack slaves master

THE PACKER ALGORITHM
EXPANSION
GEOMETRY PROPAGATION
RESTRICTIONS ON MASTER WINDOWS
PACKING ORDER
EXAMPLE
pack - Geometry manager that packs around edges of cavity

SYNOPSIS

pack option arg ?arg ...?

DESCRIPTION

The pack command is used to communicate with the packer, a geometry manager that arranges the children of a parent by packing them in order around the edges of the parent. The pack command can have any of several forms, depending on the option argument:

pack slave ?slave ...? ?options?
    If the first argument to pack is a window name (any value starting with "."), then the command is processed in the same way as pack configure.

pack configure slave ?slave ...? ?options?
    The arguments consist of the names of one or more slave windows followed by pairs of arguments that specify how to manage the slaves. See THE PACKER ALGORITHM below for details on how the options are used by the packer. The following options are supported:

    -after other
        Other must the name of another window. Use its master as the master for the slaves, and insert the slaves just after other in the packing order.

    -anchor anchor
        Anchor must be a valid anchor position such as n or sw; it specifies where to position each slave in its parcel. Defaults to
-before other

Other must the name of another window. Use its master as the master for the slaves, and insert the slaves just before other in the packing order.

-expand boolean

Specifies whether the slaves should be expanded to consume extra space in their master. Boolean may have any proper boolean value, such as 1 or no. Defaults to 0.

-fill style

If a slave's parcel is larger than its requested dimensions, this option may be used to stretch the slave. Style must have one of the following values:

none

Give the slave its requested dimensions plus any internal padding requested with -ipadx or -ipady. This is the default.

x

Stretch the slave horizontally to fill the entire width of its parcel (except leave external padding as specified by -padx).

y

Stretch the slave vertically to fill the entire height of its parcel (except leave external padding as specified by -pady).

both

Stretch the slave both horizontally and vertically.

-in other

Insert the slave(s) at the end of the packing order for the master window given by other.
-ipadx amount
   Amount specifies how much horizontal internal padding to leave on each side of the slave(s). Amount must be a valid screen distance, such as 2 or .5c. It defaults to 0.

-ipady amount
   Amount specifies how much vertical internal padding to leave on each side of the slave(s). Amount defaults to 0.

-padx amount
   Amount specifies how much horizontal external padding to leave on each side of the slave(s). Amount may be a list of two values to specify padding for left and right separately. Amount defaults to 0.

-pady amount
   Amount specifies how much vertical external padding to leave on each side of the slave(s). Amount may be a list of two values to specify padding for top and bottom separately. Amount defaults to 0.

-side side
   Specifies which side of the master the slave(s) will be packed against. Must be left, right, top, or bottom. Defaults to top.

If no -in, -after or -before option is specified then each of the slaves will be inserted at the end of the packing list for its parent unless it is already managed by the packer (in which case it will be left where it is). If one of these options is specified then all the slaves will be inserted at the specified point. If any of the slaves are already managed by the geometry manager then any unspecified options for them retain their previous values rather than receiving default values.

pack forget slave ?slave ...?
   Removes each of the slaves from the packing order for its master and unmaps their windows. The slaves will no longer be managed by the packer.
**pack info slave**
Returns a list whose elements are the current configuration state of the slave given by slave in the same option-value form that might be specified to **pack configure**. The first two elements of the list are “-in master” where master is the slave's master.

**pack propagate master ?boolean?**
If boolean has a true boolean value such as 1 or on then propagation is enabled for master, which must be a window name (see GEOMETRY PROPAGATION below). If boolean has a false boolean value then propagation is disabled for master. In either of these cases an empty string is returned. If boolean is omitted then the command returns 0 or 1 to indicate whether propagation is currently enabled for master. Propagation is enabled by default.

**pack slaves master**
Returns a list of all of the slaves in the packing order for master. The order of the slaves in the list is the same as their order in the packing order. If master has no slaves then an empty string is returned.

---

**THE PACKER ALGORITHM**

For each master the packer maintains an ordered list of slaves called the **packing list**. The -in, -after, and -before configuration options are used to specify the master for each slave and the slave's position in the packing list. If none of these options is given for a slave then the slave is added to the end of the packing list for its parent.

The packer arranges the slaves for a master by scanning the packing list in order. At the time it processes each slave, a rectangular area within the master is still unallocated. This area is called the **cavity**; for the first slave it is the entire area of the master.

For each slave the packer carries out the following steps:

[1] The packer allocates a rectangular parcel for the slave along the
side of the cavity given by the slave's -side option. If the side is top or bottom then the width of the parcel is the width of the cavity and its height is the requested height of the slave plus the -ipady and -pady options. For the left or right side the height of the parcel is the height of the cavity and the width is the requested width of the slave plus the -ipadx and -padx options. The parcel may be enlarged further because of the -expand option (see EXPANSION below)

[2]
The packer chooses the dimensions of the slave. The width will normally be the slave's requested width plus twice its -ipadx option and the height will normally be the slave's requested height plus twice its -ipady option. However, if the -fill option is x or both then the width of the slave is expanded to fill the width of the parcel, minus twice the -padx option. If the -fill option is y or both then the height of the slave is expanded to fill the width of the parcel, minus twice the -pady option.

[3]
The packer positions the slave over its parcel. If the slave is smaller than the parcel then the -anchor option determines where in the parcel the slave will be placed. If -padx or -pady is non-zero, then the given amount of external padding will always be left between the slave and the edges of the parcel.

Once a given slave has been packed, the area of its parcel is subtracted from the cavity, leaving a smaller rectangular cavity for the next slave. If a slave does not use all of its parcel, the unused space in the parcel will not be used by subsequent slaves. If the cavity should become too small to meet the needs of a slave then the slave will be given whatever space is left in the cavity. If the cavity shrinks to zero size, then all remaining slaves on the packing list will be unmapped from the screen until the master window becomes large enough to hold them again.

**EXPANSION**
If a master window is so large that there will be extra space left over after all of its slaves have been packed, then the extra space is distributed uniformly among all of the slaves for which the `-expand` option is set. Extra horizontal space is distributed among the expandable slaves whose `-side` is `left` or `right`, and extra vertical space is distributed among the expandable slaves whose `-side` is `top` or `bottom`.

**GEOMETRY PROPAGATION**

The packer normally computes how large a master must be to just exactly meet the needs of its slaves, and it sets the requested width and height of the master to these dimensions. This causes geometry information to propagate up through a window hierarchy to a top-level window so that the entire sub-tree sizes itself to fit the needs of the leaf windows. However, the `pack propagate` command may be used to turn off propagation for one or more masters. If propagation is disabled then the packer will not set the requested width and height of the packer. This may be useful if, for example, you wish for a master window to have a fixed size that you specify.

**REstrictions on master windows**

The master for each slave must either be the slave's parent (the default) or a descendant of the slave's parent. This restriction is necessary to guarantee that the slave can be placed over any part of its master that is visible without danger of the slave being clipped by its parent.

**PACKING ORDER**

If the master for a slave is not its parent then you must make sure that the slave is higher in the stacking order than the master. Otherwise the master will obscure the slave and it will appear as if the slave has not been packed correctly. The easiest way to make sure the slave is higher than the master is to create the master window first: the most recently created window will be highest in the stacking order. Or, you
can use the `raise` and `lower` commands to change the stacking order of either the master or the slave.

**EXAMPLE**

```bash
# Make the widgets
label .t -text "This widget is at the top" -bg red
label .b -text "This widget is at the bottom" -bg green
label .l -text "Left\nHand\nSide"
label .r -text "Right\nHand\nSide"
text .mid
    .mid insert end "This layout is like Java's BorderLayout"
# Lay them out
pack .t -side top -fill x
pack .b -side bottom -fill x
pack .l -side left -fill y
pack .r -side right -fill y
pack .mid -expand 1 -fill both
```

**SEE ALSO**

`grid`, `place`

**KEYWORDS**

`geometry manager`, `location`, `packer`, `parcel`, `propagation`, `size`
NAME

tk_chooseDirectory - pops up a dialog box for the user to select a directory.

SYNOPSIS

DESCRIPTION

-\texttt{-initialdir} \texttt{dirname}
-\texttt{-mustexist} \texttt{boolean}
-\texttt{-parent} \texttt{window}
-\texttt{-title} \texttt{titleString}

EXAMPLE

SEE ALSO

KEYWORDS

NAME

\texttt{tk\_chooseDirectory} - pops up a dialog box for the user to select a directory.

SYNOPSIS

\texttt{tk\_chooseDirectory} \texttt{?option value ...?}

DESCRIPTION

The procedure \texttt{tk\_chooseDirectory} pops up a dialog box for the user to select a directory. The following \texttt{option-value} pairs are possible as command line arguments:

-\texttt{-initialdir} \texttt{dirname}
  Specifies that the directories in \texttt{directory} should be displayed when the dialog pops up. If this parameter is not specified, then the directories in the current working directory are displayed. If the
parameter specifies a relative path, the return value will convert the relative path to an absolute path.

-mustexist boolean
Specifies whether the user may specify non-existent directories. If this parameter is true, then the user may only select directories that already exist. The default value is false.

-parent window
Makes window the logical parent of the dialog. The dialog is displayed on top of its parent window. On Mac OS X, this turns the file dialog into a sheet attached to the parent window.

-title titleString
Specifies a string to display as the title of the dialog box. If this option is not specified, then a default title will be displayed.

EXAMPLE

```plaintext
set dir [tk_chooseDirectory \ 
   -initialdir ~ -title "Choose a directory"]
if {$dir eq ""} {
   label .l -text "No directory selected"
} else {
   label .l -text "Selected $dir"
}
```

SEE ALSO

tk_getOpenFile, tk_getSaveFile

KEYWORDS

directory, selection, dialog, platform-specific
NAME

tkerror - Command invoked to process background errors

SYNOPSIS

tkerror message

DESCRIPTION

Note: as of Tk 4.1 the tkerror command has been renamed to bgerror because the event loop (which is what usually invokes it) is now part of Tcl. For backward compatibility the bgerror provided by the current Tk version still tries to call tkerror if there is one (or an auto loadable one), so old script defining that error handler should still work, but you should anyhow modify your scripts to use bgerror instead of tkerror because that support for the old name might vanish in the near future. If that call fails, bgerror posts a dialog showing the error and offering to see the stack trace to the user. If you want your own error management you should directly override bgerror instead of tkerror. Documentation for bgerror is available as part of Tcl's documentation.

KEYWORDS

background error, reporting
NAME

ttk::progressbar - Provide progress feedback

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS

-class
-cursor, cursor, Cursor
-style
-takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS

-orient, orient, Orient
-length, length, Length
-mode, mode, Mode
-maximum, maximum, Maximum
-value, value, Value
-variable, variable, Variable
-phase, phase, Phase

WIDGET COMMAND

pathName cget option
pathName configure ?option? ?value option value ...?
pathName identify x y
pathName instate statespec ?script?
pathName start ?interval?
pathName state ?stateSpec?
pathName step ?amount?
pathName stop

SEE ALSO

NAME

ttk::progressbar - Provide progress feedback
SYNOPSIS

```plaintext
ttk::progressbar pathName ?options?
```

DESCRIPTION

A `ttk::progressbar` widget shows the status of a long-running operation. They can operate in two modes: `determinate` mode shows the amount completed relative to the total amount of work to be done, and `indeterminate` mode provides an animated display to let the user know that something is happening.

STANDARD OPTIONS

- `class`
- `cursor, cursor, Cursor`
- `style`
- `takefocus, takeFocus, TakeFocus`

WIDGET-SPECIFIC OPTIONS

Command-Line Name: `-orient`
Database Name: `orient`
Database Class: `Orient`
    One of `horizontal` or `vertical`. Specifies the orientation of the progress bar.

Command-Line Name: `-length`
Database Name: `length`
Database Class: `Length`
    Specifies the length of the long axis of the progress bar (width if horizontal, height if vertical).

Command-Line Name: `-mode`
Database Name: `mode`
Database Class: `Mode`
    One of `determinate` or `indeterminate`. 
Command-Line Name: **-maximum**
Database Name: **maximum**
Database Class: **Maximum**
   A floating point number specifying the maximum **-value**. Defaults to 100.

Command-Line Name: **-value**
Database Name: **value**
Database Class: **Value**
   The current value of the progress bar. In *determinate* mode, this represents the amount of work completed. In *indeterminate* mode, it is interpreted modulo **-maximum**; that is, the progress bar completes one “cycle” when the **-value** increases by **-maximum**.

Command-Line Name: **-variable**
Database Name: **variable**
Database Class: **Variable**
   The name of a Tcl variable which is linked to the **-value**. If specified, the **-value** of the progress bar is automatically set to the value of the variable whenever the latter is modified.

Command-Line Name: **-phase**
Database Name: **phase**
Database Class: **Phase**
   Read-only option. The widget periodically increments the value of this option whenever the **-value** is greater than 0 and, in *determinate* mode, less than **-maximum**. This option may be used by the current theme to provide additional animation effects.

**WIDGET COMMAND**

```tcl
pathName cget option
   Returns the current value of the specified option; see `ttk::widget(n)`.

pathName configure ?option? ?value option value ...? 
   Modify or query widget options; see `ttk::widget(n)`.

pathName identify x y
```
Returns the name of the element at position $x, y$. See `ttk::widget(n)`.

`pathName instate stateSpec ?script?`
Test the widget state; see `ttk::widget(n)`.

`pathName start ?interval?`
Begin autoincrement mode: schedules a recurring timer event that calls `step` every `interval` milliseconds. If omitted, `interval` defaults to 50 milliseconds (20 steps/second).

`pathName state ?stateSpec?`
Modify or query the widget state; see `ttk::widget(n)`.

`pathName step ?amount?`
Increments the `-value` by `amount`. `amount` defaults to 1.0 if omitted.

`pathName stop`
Stop autoincrement mode: cancels any recurring timer event initiated by `pathName start`.

**SEE ALSO**

`ttk::widget`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2005 Joe English
NAME

bindtags - Determine which bindings apply to a window, and order of evaluation

SYNOPSIS

bindtags window ?tagList?

DESCRIPTION

When a binding is created with the **bind** command, it is associated either with a particular window such as `.a.b.c`, a class name such as **Button**, the keyword **all**, or any other string. All of these forms are called *binding tags*. Each window contains a list of binding tags that determine how events are processed for the window. When an event occurs in a window, it is applied to each of the window's tags in order: for each tag, the most specific binding that matches the given tag and event is executed. See the **bind** command for more information on the matching process.

By default, each window has four binding tags consisting of the name of the window, the window's class name, the name of the window's nearest toplevel ancestor, and **all**, in that order. Toplevel windows have only three tags by default, since the toplevel name is the same as that of the window. The **bindtags** command allows the binding tags for a window to be read and modified.

If **bindtags** is invoked with only one argument, then the current set of binding tags for *window* is returned as a list. If the *tagList* argument is specified to **bindtags**, then it must be a proper list; the tags for *window* are changed to the elements of the list. The elements of *tagList* may be
arbitrary strings; however, any tag starting with a dot is treated as the name of a window; if no window by that name exists at the time an event is processed, then the tag is ignored for that event. The order of the elements in tagList determines the order in which binding scripts are executed in response to events. For example, the command

```
bindtags .b {all . Button .b}
```

reverses the order in which binding scripts will be evaluated for a button named .b so that all bindings are invoked first, following by bindings for .b's toplevel ("."), followed by class bindings, followed by bindings for .b. If tagList is an empty list then the binding tags for window are returned to the default state described above.

The bindtags command may be used to introduce arbitrary additional binding tags for a window, or to remove standard tags. For example, the command

```
bindtags .b { .b TrickyButton . all}
```

replaces the Button tag for .b with TrickyButton. This means that the default widget bindings for buttons, which are associated with the Button tag, will no longer apply to .b, but any bindings associated with TrickyButton (perhaps some new button behavior) will apply.

**EXAMPLE**

If you have a set of nested frame widgets and you want events sent to a button widget to also be delivered to all the widgets up to the current toplevel (in contrast to Tk's default behavior, where events are not delivered to those intermediate windows) to make it easier to have accelerators that are only active for part of a window, you could use a helper procedure like this to help set things up:
proc setupBindtagsForTreeDelivery {widget} {
    set tags [list $widget [winfo class $widget]]
    set w $widget
    set t [winfo toplevel $w]
    while {$w ne $t} {
        set w [winfo parent $w]
        lappend tags $w
    }
    lappend tags all
    bindtags $widget $tags
}

SEE ALSO

bind

KEYWORDS

binding, event, tag

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
    grab - Confine pointer and keyboard events to a window sub-tree

SYNOPSIS

DESCRIPTION

    grab ?-global? window
    grab current ?window?
    grab release window
    grab set ?-global? window
    grab status window

WARNING

BUGS

EXAMPLE

KEYWORDS

NAME

grab - Confine pointer and keyboard events to a window sub-tree

SYNOPSIS

grab ?-global? window
grab option ?arg arg ...?

DESCRIPTION

This command implements simple pointer and keyboard grabs for Tk. Tk's grabs are different than the grabs described in the Xlib documentation. When a grab is set for a particular window, Tk restricts all pointer events to the grab window and its descendants in Tk's window hierarchy. Whenever the pointer is within the grab window's subtree, the pointer will behave exactly the same as if there had been
no grab at all and all events will be reported in the normal fashion. When the pointer is outside window's tree, button presses and releases and mouse motion events are reported to window, and window entry and window exit events are ignored. The grab subtree “owns” the pointer: windows outside the grab subtree will be visible on the screen but they will be insensitive until the grab is released. The tree of windows underneath the grab window can include top-level windows, in which case all of those top-level windows and their descendants will continue to receive mouse events during the grab.

Two forms of grabs are possible: local and global. A local grab affects only the grabbing application: events will be reported to other applications as if the grab had never occurred. Grabs are local by default. A global grab locks out all applications on the screen, so that only the given subtree of the grabbing application will be sensitive to pointer events (mouse button presses, mouse button releases, pointer motions, window entries, and window exits). During global grabs the window manager will not receive pointer events either.

During local grabs, keyboard events (key presses and key releases) are delivered as usual: the window manager controls which application receives keyboard events, and if they are sent to any window in the grabbing application then they are redirected to the focus window. During a global grab Tk grabs the keyboard so that all keyboard events are always sent to the grabbing application. The focus command is still used to determine which window in the application receives the keyboard events. The keyboard grab is released when the grab is released.

Grabs apply to particular displays. If an application has windows on multiple displays then it can establish a separate grab on each display. The grab on a particular display affects only the windows on that display. It is possible for different applications on a single display to have simultaneous local grabs, but only one application can have a global grab on a given display at once.

The grab command can take any of the following forms:
**grab** ?-global? *window*
   Same as **grab set**, described below.

**grab current** ?*window*?
   If *window* is specified, returns the name of the current grab window in this application for *window*'s display, or an empty string if there is no such window. If *window* is omitted, the command returns a list whose elements are all of the windows grabbed by this application for all displays, or an empty string if the application has no grabs.

**grab release** *window*
   Releases the grab on *window* if there is one, otherwise does nothing. Returns an empty string.

**grab set** ?-global? *window*
   Sets a grab on *window*. If -global is specified then the grab is global, otherwise it is local. If a grab was already in effect for this application on *window*'s display then it is automatically released. If there is already a grab on *window* and it has the same global/local form as the requested grab, then the command does nothing. Returns an empty string.

**grab status** *window*
   Returns **none** if no grab is currently set on *window*, **local** if a local grab is set on *window*, and **global** if a global grab is set.

---

**WARNING**

It is very easy to use global grabs to render a display completely unusable (e.g. by setting a grab on a widget which does not respond to events and not providing any mechanism for releasing the grab). Take **extreme** care when using them!

---

**BUGS**

It took an incredibly complex and gross implementation to produce the simple grab effect described above. Given the current implementation, it is not safe for applications to use the Xlib grab facilities at all except
through the Tk grab procedures. If applications try to manipulate X's grab mechanisms directly, things will probably break.

If a single process is managing several different Tk applications, only one of those applications can have a local grab for a given display at any given time. If the applications are in different processes, this restriction does not exist.

**EXAMPLE**

Set a grab so that only one button may be clicked out of a group. The other buttons are unresponsive to the mouse until the middle button is clicked.

```python
pack [button .b1 -text "Click me! #1" -command {destroy .b1}]
pack [button .b2 -text "Click me! #2" -command {destroy .b2}]
pack [button .b3 -text "Click me! #3" -command {destroy .b3}]
grab .b2
```

**KEYWORDS**

grab, keyboard events, pointer events, window

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1992 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
panedwindow - Create and manipulate panedwindow widgets

SYNOPSIS

STANDARD OPTIONS
- -background or -bg, background, Background
- -borderwidth or -bd, borderWidth, BorderWidth
- -cursor, cursor, Cursor
- -orient, orient, Orient
- -relief, relief, Relief

WIDGET-SPECIFIC OPTIONS
- -handlepad, handlePad, HandlePad
- -handlesize, handleSize, HandleSize
- -height, height, Height
- -opaqueresize, opaqueResize, OpaqueResize
- -sashcursor, sashCursor, SashCursor
- -sashpad, sashPad, SashPad
- -sashrelief, sashRelief, SashRelief
- -sashwidth, sashWidth, SashWidth
- -showhandle, showHandle, ShowHandle
- -width, width, Width

DESCRIPTION

WIDGET COMMAND

pathName add window ?window ...? ?option value ...?
pathName cget option
pathName configure ?option? ?value option value ...?
pathName forget window ?window ...?
pathName identify x y
pathName proxy ?args?

pathName proxy coord
pathName proxy forget
pathName proxy place x y
**NAME**

panedwindow - Create and manipulate panedwindow widgets

**SYNOPSIS**

```
panedwindow pathName ?options?
```

**STANDARD OPTIONS**

- **pathName sash ?args?**
  - **pathName sash coord index**
  - **pathName sash dragto index x y**
  - **pathName sash mark index x y**
  - **pathName sash place index x y**

- **pathName panecget window option**

- **pathName paneconfigure window ?option? ?value option value ...?**
  - `-after window`
  - `-before window`
  - `-height size`
  - `-hide boolean`
  - `-minsize n`
  - `-padx n`
  - `-pady n`
  - `-sticky style`
  - `-stretch when`
    - `always`
    - `first`
    - `last`
    - `middle`
    - `never`
  - `-width size`

- **pathName panes**

RESIZING PANES
SEE ALSO
KEYWORDS
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-orient, orient, Orient
-relief, relief, Relief

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: **-handlepad**
Database Name: **handlePad**
Database Class: **HandlePad**
   When sash handles are drawn, specifies the distance from the top or left end of the sash (depending on the orientation of the widget) at which to draw the handle. May be any value accepted by **Tk_GetPixels**.

Command-Line Name: **-handlesize**
Database Name: **handleSize**
Database Class: **HandleSize**
   Specifies the side length of a sash handle. Handles are always drawn as squares. May be any value accepted by **Tk_GetPixels**.

Command-Line Name: **-height**
Database Name: **height**
Database Class: **Height**
   Specifies a desired height for the overall panedwindow widget. May be any value accepted by **Tk_GetPixels**. If an empty string, the widget will be made high enough to allow all contained widgets to have their natural height.

Command-Line Name: **-opaqueresize**
Database Name: **opaqueResize**
Database Class: **OpaqueResize**
   Specifies whether panes should be resized as a sash is moved (true), or if resizing should be deferred until the sash is placed (false).

Command-Line Name: **-sashcursor**
Database Name: **sashCursor**  
**Mouse cursor to use when over a sash. If null, `sb_h_double_arrow` will be used for horizontal panedwindows, and `sb_v_double_arrow` will be used for vertical panedwindows.**

Command-Line Name: **-sashpad**  
Database Name: **sashPad**  
**Specifies the amount of padding to leave of each side of a sash. May be any value accepted by `Tk_GetPixels`.**

Command-Line Name: **-sashrelief**  
Database Name: **sashRelief**  
**Relief to use when drawing a sash. May be any of the standard Tk relief values.**

Command-Line Name: **-sashwidth**  
Database Name: **sashWidth**  
**Specifies the width of each sash. May be any value accepted by `Tk_GetPixels`.**

Command-Line Name: **-showhandle**  
Database Name: **showHandle**  
**Specifies whether sash handles should be shown. May be any valid Tcl boolean value.**

Command-Line Name: **-width**  
Database Name: **width**  
**Specifies a desired width for the overall panedwindow widget. May be any value accepted by `Tk_GetPixels`. If an empty string, the widget will be made wide enough to allow all contained widgets to have their natural width.**
The **panedwindow** command creates a new window (given by the *pathName* argument) and makes it into a panedwindow widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the panedwindow such as its default background color and relief. The **panedwindow** command returns the path name of the new window.

A panedwindow widget contains any number of panes, arranged horizontally or vertically, according to the value of the **-orient** option. Each pane contains one widget, and each pair of panes is separated by a moveable (via mouse movements) sash. Moving a sash causes the widgets on either side of the sash to be resized.

**WIDGET COMMAND**

The **panedwindow** command creates a new Tcl command whose name is the same as the path name of the panedwindow's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
    pathName option ?arg arg ...? ?
```

*PathName* is the name of the command, which is the same as the panedwindow widget's path name. *Option* and the *args* determine the exact behavior of the command. The following commands are possible for panedwindow widgets:

```
    pathName add window ?window ...? ?option value ...?
```

Add one or more windows to the panedwindow, each in a separate pane. The arguments consist of the names of one or more windows followed by pairs of arguments that specify how to manage the windows. *Option* may have any of the values accepted by the **configure** subcommand.
**pathName** cget **option**

Returns the current value of the configuration option given by **option**. **Option** may have any of the values accepted by the **panedwindow** command.

**pathName** configure ?**option**? ?**value** option value ...?

Query or modify the configuration options of the widget. If no **option** is specified, returns a list describing all of the available options for **pathName** (see **Tk_ConfigureInfo** for information on the format of this list). If **option** is specified with no **value**, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no **option** is specified). If one or more **option-value** pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. **Option** may have any of the values accepted by the **panedwindow** command.

**pathName** forget **window** ?**window** ...?

Remove the pane containing **window** from the panedwindow. All geometry management options for **window** will be forgotten.

**pathName** identify x y

Identify the panedwindow component underneath the point given by x and y, in window coordinates. If the point is over a sash or a sash handle, the result is a two element list containing the index of the sash or handle, and a word indicating whether it is over a sash or a handle, such as {0 sash} or {2 handle}. If the point is over any other part of the panedwindow, the result is an empty list.

**pathName** proxy ?**args**?

This command is used to query and change the position of the sash proxy, used for rubberband-style pane resizing. It can take any of the following forms:

**pathName** proxy coord

Return a list containing the x and y coordinates of the most recent proxy location.
**pathName proxy forget**  
Remove the proxy from the display.

**pathName proxy place x y**  
Place the proxy at the given x and y coordinates.

**pathName sash ?args?**  
This command is used to query and change the position of sashes in the panedwindow. It can take any of the following forms:

**pathName sash coord index**  
Return the current x and y coordinate pair for the sash given by index. Index must be an integer between 0 and 1 less than the number of panes in the panedwindow. The coordinates given are those of the top left corner of the region containing the sash.

**pathName sash dragto index x y**  
This command computes the difference between the given coordinates and the coordinates given to the last sash mark command for the given sash. It then moves that sash the computed difference. The return value is the empty string.

**pathName sash mark index x y**  
Records x and y for the sash given by index; used in conjunction with later sash dragto commands to move the sash.

**pathName sash place index x y**  
Place the sash given by index at the given coordinates.

**pathName paneconfigure window ?option? ?value option value ...?**  
Query or modify the management options for window. If no option is specified, returns a list describing all of the available options for
pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option-value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. The following options are supported:

-**after** window
  Insert the window after the window specified. window should be the name of a window already managed by pathName.

-**before** window
  Insert the window before the window specified. window should be the name of a window already managed by pathName.

-**height** size
  Specify a height for the window. The height will be the outer dimension of the window including its border, if any. If size is an empty string, or if -height is not specified, then the height requested internally by the window will be used initially; the height may later be adjusted by the movement of sashes in the panedwindow. Size may be any value accepted by Tk_GetPixels.

-**hide** boolean
  Controls the visibility of a pane. When the boolean is true (according to Tcl_GetBoolean) the pane will not be visible, but it will still be maintained in the list of panes.

-**minsize** n
  Specifies that the size of the window cannot be made less than n. This constraint only affects the size of the widget in the paned dimension — the x dimension for horizontal panedwindows, the y dimension for vertical panedwindows. May be any value accepted by Tk_GetPixels.
-padx \(n\)
Specifies a non-negative value indicating how much extra space to leave on each side of the window in the X-direction. The value may have any of the forms accepted by `Tk_GetPixels`.

-pady \(n\)
Specifies a non-negative value indicating how much extra space to leave on each side of the window in the Y-direction. The value may have any of the forms accepted by `Tk_GetPixels`.

-sticky \textit{style}\nIf a window's pane is larger than the requested dimensions of the window, this option may be used to position (or stretch) the window within its pane. \textit{Style} is a string that contains zero or more of the characters \texttt{n}, \texttt{s}, \texttt{e} or \texttt{w}. The string can optionally contain spaces or commas, but they are ignored. Each letter refers to a side (north, south, east, or west) that the window will “stick” to. If both \texttt{n} and \texttt{s} (or \texttt{e} and \texttt{w}) are specified, the window will be stretched to fill the entire height (or width) of its cavity.

-stretch \textit{when}\nControls how extra space is allocated to each of the panes. \textit{When} is one of \texttt{always}, \texttt{first}, \texttt{last}, \texttt{middle}, and \texttt{never}. The panedwindow will calculate the required size of all its panes. Any remaining (or deficit) space will be distributed to those panes marked for stretching. The space will be distributed based on each panes current ratio of the whole. The \textit{when} values have the following definition:

\texttt{always}\n
This pane will always stretch.

\texttt{first}\n
Only if this pane is the first pane (left-most or top-most) will it stretch.
last
   Only if this pane is the last pane (right-most or bottom-most) will it stretch. This is the default value.

middle
   Only if this pane is not the first or last pane will it stretch.

never
   This pane will never stretch.

-width size
   Specify a width for the window. The width will be the outer dimension of the window including its border, if any. If size is an empty string, or if -width is not specified, then the width requested internally by the window will be used initially; the width may later be adjusted by the movement of sashes in the panedwindow. Size may be any value accepted by Tk_GetPixels.

pathName panes
   Returns an ordered list of the widgets managed by pathName.

RESIZING PANNES

A pane is resized by grabbing the sash (or sash handle if present) and dragging with the mouse. This is accomplished via mouse motion bindings on the widget. When a sash is moved, the sizes of the panes on each side of the sash, and thus the widgets in those panes, are adjusted.

When a pane is resized from outside (e.g. it is packed to expand and fill, and the containing toplevel is resized), space is added to the final (rightmost or bottommost) pane in the window.

SEE ALSO

ttk::panedwindow
NAME
tk_dialog - Create modal dialog and wait for response

SYNOPSIS
tk_dialog

DESCRIPTION
This procedure is part of the Tk script library. Its arguments describe a dialog box:

window
Name of top-level window to use for dialog. Any existing window by this name is destroyed.

title
Text to appear in the window manager's title bar for the dialog.

text
Message to appear in the top portion of the dialog box.

bitmap
If non-empty, specifies a bitmap to display in the top portion of the dialog, to the left of the text. If this is an empty string then no bitmap is displayed in the dialog.

default
If this is an integer greater than or equal to zero, then it gives the index of the button that is to be the default button for the dialog (0 for the leftmost button, and so on). If less than zero or an empty string then there will not be any default button.

string
There will be one button for each of these arguments. Each string specifies text to display in a button, in order from left to right.

After creating a dialog box, tk_dialog waits for the user to select one of the buttons either by clicking on the button with the mouse or by typing return to invoke the default button (if any). Then it returns the index of the selected button: 0 for the leftmost button, 1 for the button next to it, and so on. If the dialog's window is destroyed before the user selects one of the buttons, then -1 is returned.

While waiting for the user to respond, tk_dialog sets a local grab. This prevents the user from interacting with the application in any way except to invoke the dialog box.

**EXAMPLE**

```
set reply [tk_dialog .foo "The Title" "Do you want to
questhead 0 Yes No "I'm not sure"]
```
SEE ALSO

tk_messageBox

KEYWORDS

bitmap, dialog, modal

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1992 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

tkvars - Variables used or set by Tk

DESCRIPTION

The following Tcl variables are either set or used by Tk at various times in its execution:

**tk_library**

This variable holds the file name for a directory containing a library of Tcl scripts related to Tk. These scripts include an initialization file that is normally processed whenever a Tk application starts up, plus other files containing procedures that implement default behaviors for widgets. The initial value of tk_library is set when Tk is added to an interpreter; this is done by searching several different directories until one is found that contains an appropriate Tk startup script. If the TK_LIBRARY environment variable exists, then the directory it names is checked first. If TK_LIBRARY is not set or does not refer to an appropriate directory, then Tk checks several other directories based on a compiled-in default location, the location of the Tcl library directory, the location of the binary containing the application, and the current working directory. The variable can be modified by an application to switch to a different library.

**tk_patchLevel**

Contains a decimal integer giving the current patch level for Tk. The patch level is incremented for each new release or patch, and it uniquely identifies an official version of Tk.

**tk::Priv**
This variable is an array containing several pieces of information that are private to Tk. The elements of `tk::Priv` are used by Tk library procedures and default bindings. They should not be accessed by any code outside Tk.

**tk_strictMotif**

This variable is set to zero by default. If an application sets it to one, then Tk attempts to adhere as closely as possible to Motif look-and-feel standards. For example, active elements such as buttons and scrollbar sliders will not change color when the pointer passes over them.

**tk_textRedraw**

**tk_textRelayout**

These variables are set by text widgets when they have debugging turned on. The values written to these variables can be used to test or debug text widget operations. These variables are mostly used by Tk's test suite.

**tk_version**

Tk sets this variable in the interpreter for each application. The variable holds the current version number of the Tk library in the form `major.minor`. `Major` and `minor` are integers. The major version number increases in any Tk release that includes changes that are not backward compatible (i.e. whenever existing Tk applications and scripts may have to change to work with the new release). The minor version number increases with each new release of Tk, except that it resets to zero whenever the major version number changes.

---

**KEYWORDS**

variables, version, text

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
**NAME**

```
ttk::radiobutton - Mutually exclusive option widget
```

**SYNOPSIS**

**DESCRIPTION**

**STANDARD OPTIONS**

- `-class`  
- `-compound, compound, Compound`
- `-cursor, cursor, Cursor`
- `-image, image, Image`
- `-state`  
- `-style`  
- `-takefocus, takeFocus, TakeFocus`
- `-text, text, Text`
- `-textvariable, textVariable, Variable`
- `-underline, underline, Underline`
- `-width`

**WIDGET-SPECIFIC OPTIONS**

- `-command, command, Command`
- `-value, Value, Value`
- `-variable, variable, Variable`

**WIDGET COMMAND**

```
pathname invoke
```

**WIDGET STATES**

**SEE ALSO**

**KEYWORDS**

**NAME**

```
ttk::radiobutton - Mutually exclusive option widget
```

**SYNOPSIS**
**ttk::radiobutton** *pathName* ?*options*?

### DESCRIPTION

*ttk::radiobutton* widgets are used in groups to show or change a set of mutually-exclusive options. Radiobuttons are linked to a Tcl variable, and have an associated value; when a radiobutton is clicked, it sets the variable to its associated value.

### STANDARD OPTIONS

- `-class`
- `-compound`, `compound`, `Compound`
- `-cursor`, `cursor`, `Cursor`
- `-image`, `image`, `Image`
- `-state`
- `-style`
- `-takefocus`, `takeFocus`, `TakeFocus`
- `-text`, `text`, `Text`
- `-textvariable`, `textVariable`, `Variable`
- `-underline`, `underline`, `Underline`
- `-width`

### WIDGET-SPECIFIC OPTIONS

Command-Line Name: `-command`
Database Name: `command`
Database Class: `Command`
  A Tcl script to evaluate whenever the widget is invoked.

Command-Line Name: `-value`
Database Name: `Value`
Database Class: `Value`
  The value to store in the associated `-variable` when the widget is selected.

Command-Line Name: `-variable`
Database Name: `variable`
Database Class: **Variable**

The name of a global variable whose value is linked to the widget. Default value is `::selectedButton`.

### WIDGET COMMAND

In addition to the standard `cget`, `configure`, `identify`, `instate`, and `state` commands, radiobuttons support the following additional widget commands:

**pathname invoke**

Sets the `-variable` to the `-value`, selects the widget, and evaluates the associated `-command`. Returns the result of the `-command`, or the empty string if no `-command` is specified.

### WIDGET STATES

The widget does not respond to user input if the `disabled` state is set. The widget sets the `selected` state whenever the linked `-variable` is set to the widget's `-value`, and clears it otherwise. The widget sets the `alternate` state whenever the linked `-variable` is unset. (The `alternate` state may be used to indicate a “tri-state” or “indeterminate” selection.)

### SEE ALSO

`ttk::widget`, `ttk::checkbutton`, `radiobutton`

### KEYWORDS

`widget`, `button`, `option`

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2004 Joe English
NAME

bitmap - Images that display two colors

SYNOPSIS

DESCRIPTION

CREATING BITMAPS

-background color
-data string
-file name
-foreground color
-maskdata string
-maskfile name

IMAGE COMMAND

imageName cget option
imageName configure ?option? ?value option value ...?

KEYWORDS
transparent effect. For other pixels, the image displays the foreground color if the source data is one and the background color if the source data is zero.

## CREATING BITMAPS

Like all images, bitmaps are created using the `image create` command. Bitmaps support the following options:

- **background color**
  Specifies a background color for the image in any of the standard ways accepted by Tk. If this option is set to an empty string then the background pixels will be transparent. This effect is achieved by using the source bitmap as the mask bitmap, ignoring any **-maskdata** or **-maskfile** options.

- **data string**
  Specifies the contents of the source bitmap as a string. The string must adhere to X11 bitmap format (e.g., as generated by the `bitmap` program). If both the **-data** and **-file** options are specified, the **-data** option takes precedence.

- **file name**
  `name` gives the name of a file whose contents define the source bitmap. The file must adhere to X11 bitmap format (e.g., as generated by the `bitmap` program).

- **foreground color**
  Specifies a foreground color for the image in any of the standard ways accepted by Tk.

- **maskdata string**
  Specifies the contents of the mask as a string. The string must adhere to X11 bitmap format (e.g., as generated by the `bitmap` program). If both the **-maskdata** and **-maskfile** options are specified, the **-maskdata** option takes precedence.

- **maskfile name**
*name* gives the name of a file whose contents define the mask. The file must adhere to X11 bitmap format (e.g., as generated by the *bitmap* program).

**IMAGE COMMAND**

When a bitmap image is created, Tk also creates a new command whose name is the same as the image. This command may be used to invoke various operations on the image. It has the following general form:

```
imageName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for bitmap images:

```
imageName cget option
```

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the *image create bitmap* command.

```
imageName configure ?option? ?value option value ...?
```

Query or modify the configuration options for the image. If no *option* is specified, returns a list describing all of the available options for *imageName* (see *Tk_ConfigureInfo* for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option-value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the *image create bitmap* command.

**KEYWORDS**
bitmap, image

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

grid - Geometry manager that arranges widgets in a grid

SYNOPSIS

DESCRIPTION

grid slave ?slave ...? ?options?
grid anchor master ?anchor?
grid bbox master ?column row? ?column2 row2?
grid columnconfigure master index ?-option value...?
grid configure slave ?slave ...? ?options?
  -column n
  -columnspan n
  -in other
  -ipadx amount
  -ipady amount
  -padx amount
  -pady amount
  -row n
  -rowspan n
  -sticky style

grid forget slave ?slave ...?
grid info slave
grid location master x y
grid propagate master ?boolean?
grid rowconfigure master index ?-option value...?
grid remove slave ?slave ...?
grid size master
grid slaves master ?-option value?

RELATIVE PLACEMENT

- x
  ^
grid - Geometry manager that arranges widgets in a grid

grid option arg ?arg ...?

The `grid` command is used to communicate with the grid geometry manager that arranges widgets in rows and columns inside of another window, called the geometry master (or master window). The `grid` command can have any of several forms, depending on the `option` argument:

```
grid slave ?slave ...? ?options?
```

If the first argument to `grid` is suitable as the first slave argument to `grid configure`, either a window name (any value starting with .) or one of the characters x or ^ (see the RELATIVE PLACEMENT section below), then the command is processed in the same way as `grid configure`.

```
grid anchor master ?anchor?
```

The anchor value controls how to place the grid within the master when no row/column has any weight. See THE GRID ALGORITHM below for further details. The default anchor is nw.

```
grid bbox master ?column row? ?column2 row2?
```
With no arguments, the bounding box (in pixels) of the grid is returned. The return value consists of 4 integers. The first two are the pixel offset from the master window (x then y) of the top-left corner of the grid, and the second two integers are the width and height of the grid, also in pixels. If a single column and row is specified on the command line, then the bounding box for that cell is returned, where the top left cell is numbered from zero. If both column and row arguments are specified, then the bounding box spanning the rows and columns indicated is returned.

grid columnconfigure master index ?-option value...? Query or set the column properties of the index column of the geometry master, master. The valid options are -minsize, -weight, -uniform and -pad. If one or more options are provided, then index may be given as a list of column indices to which the configuration options will operate on. Indices may be integers, window names or the keyword all. For all the options apply to all columns currently occupied be slave windows. For a window name, that window must be a slave of this master and the options apply to all columns currently occupied be the slave. The -minsize option sets the minimum size, in screen units, that will be permitted for this column. The -weight option (an integer value) sets the relative weight for apportioning any extra spaces among columns. A weight of zero (0) indicates the column will not deviate from its requested size. A column whose weight is two will grow at twice the rate as a column of weight one when extra space is allocated to the layout. The -uniform option, when a non-empty value is supplied, places the column in a uniform group with other columns that have the same value for -uniform. The space for columns belonging to a uniform group is allocated so that their sizes are always in strict proportion to their -weight values. See THE GRID ALGORITHM below for further details. The -pad option specifies the number of screen units that will be added to the largest window contained completely in that column when the grid geometry manager requests a size from the containing window. If only an option is specified, with no value, the current value of that option is returned. If only the master window and index is specified, all the current
settings are returned in a list of “-option value” pairs.

`grid configure slave ?slave ...? ?options?`  
The arguments consist of the names of one or more slave windows followed by pairs of arguments that specify how to manage the slaves. The characters -, x and ^, can be specified instead of a window name to alter the default location of a `slave`, as described in the RELATIVE PLACEMENT section, below. The following options are supported:

- `-column n`  
  Insert the slave so that it occupies the n th column in the grid. Column numbers start with 0. If this option is not supplied, then the slave is arranged just to the right of previous slave specified on this call to `grid`, or column “0” if it is the first slave. For each x that immediately precedes the slave, the column position is incremented by one. Thus the x represents a blank column for this row in the grid.

- `-columnspan n`  
  Insert the slave so that it occupies n columns in the grid. The default is one column, unless the window name is followed by a -, in which case the columnspan is incremented once for each immediately following -.

- `-in other`  
  Insert the slave(s) in the master window given by other. The default is the first slave’s parent window.

- `-ipadx amount`  
  The amount specifies how much horizontal internal padding to leave on each side of the slave(s). This is space is added inside the slave(s) border. The amount must be a valid screen distance, such as 2 or .5c. It defaults to 0.

- `-ipady amount`  
  The amount specifies how much vertical internal padding to leave on the top and bottom of the slave(s). This space is
added inside the slave(s) border. The amount defaults to 0.

-padx amount
The amount specifies how much horizontal external padding to leave on each side of the slave(s), in screen units. Amount may be a list of two values to specify padding for left and right separately. The amount defaults to 0. This space is added outside the slave(s) border.

-pady amount
The amount specifies how much vertical external padding to leave on the top and bottom of the slave(s), in screen units. Amount may be a list of two values to specify padding for top and bottom separately. The amount defaults to 0. This space is added outside the slave(s) border.

-row n
Insert the slave so that it occupies the nth row in the grid. Row numbers start with 0. If this option is not supplied, then the slave is arranged on the same row as the previous slave specified on this call to grid, or the first unoccupied row if this is the first slave.

-rowspan n
Insert the slave so that it occupies n rows in the grid. The default is one row. If the next grid command contains ^ characters instead of slaves that line up with the columns of this slave, then the rowspan of this slave is extended by one.

-sticky style
If a slave's cell is larger than its requested dimensions, this option may be used to position (or stretch) the slave within its cell. Style is a string that contains zero or more of the characters n, s, e or w. The string can optionally contains spaces or commas, but they are ignored. Each letter refers to a side (north, south, east, or west) that the slave will "stick" to. If both n and s (or e and w) are specified, the slave will be stretched to fill the entire height (or width) of its cavity. The
**sticky** option subsumes the combination of **-anchor** and **-fill** that is used by **pack**. The default is "", which causes the slave to be centered in its cavity, at its requested size.

If any of the slaves are already managed by the geometry manager then any unspecified options for them retain their previous values rather than receiving default values.

**grid forget slave ?slave ...?**
Removes each of the slaves from grid for its master and unmaps their windows. The slaves will no longer be managed by the grid geometry manager. The configuration options for that window are forgotten, so that if the slave is managed once more by the grid geometry manager, the initial default settings are used.

**grid info slave**
Returns a list whose elements are the current configuration state of the slave given by slave in the same option-value form that might be specified to **grid configure**. The first two elements of the list are "-in master” where master is the slave's master.

**grid location master x y**
Given x and y values in screen units relative to the master window, the column and row number at that x and y location is returned. For locations that are above or to the left of the grid, -1 is returned.

**grid propagate master ?boolean?**
If boolean has a true boolean value such as 1 or on then propagation is enabled for master, which must be a window name (see GEOMETRY PROPAGATION below). If boolean has a false boolean value then propagation is disabled for master. In either of these cases an empty string is returned. If boolean is omitted then the command returns 0 or 1 to indicate whether propagation is currently enabled for master. Propagation is enabled by default.

**grid rowconfigure master index ?-option value...?**
Query or set the row properties of the index row of the geometry master, master. The valid options are **-minsize**, **-weight**, **-uniform**
and -pad. If one or more options are provided, then index may be given as a list of row indices to which the configuration options will operate on. Indices may be integers, window names or the keyword all. For all the options apply to all rows currently occupied be slave windows. For a window name, that window must be a slave of this master and the options apply to all rows currently occupied be the slave. The -minsize option sets the minimum size, in screen units, that will be permitted for this row. The -weight option (an integer value) sets the relative weight for apportioning any extra spaces among rows. A weight of zero (0) indicates the row will not deviate from its requested size. A row whose weight is two will grow at twice the rate as a row of weight one when extra space is allocated to the layout. The -uniform option, when a non-empty value is supplied, places the row in a uniform group with other rows that have the same value for -uniform. The space for rows belonging to a uniform group is allocated so that their sizes are always in strict proportion to their -weight values. See THE GRID ALGORITHM below for further details. The -pad option specifies the number of screen units that will be added to the largest window contained completely in that row when the grid geometry manager requests a size from the containing window. If only an option is specified, with no value, the current value of that option is returned. If only the master window and index is specified, all the current settings are returned in a list of “-option value” pairs.

grid remove slave ?slave ...?
Removes each of the slaves from grid for its master and unmaps their windows. The slaves will no longer be managed by the grid geometry manager. However, the configuration options for that window are remembered, so that if the slave is managed once more by the grid geometry manager, the previous values are retained.

grid size master
Returns the size of the grid (in columns then rows) for master. The size is determined either by the slave occupying the largest row or column, or the largest column or row with a minsize, weight, or
**pad** that is non-zero.

**grid slaves** *master* ?-option value?

If no options are supplied, a list of all of the slaves in *master* are returned, most recently manages first. *Option* can be either -row or -column which causes only the slaves in the row (or column) specified by value to be returned.

### RELATIVE PLACEMENT

The **grid** command contains a limited set of capabilities that permit layouts to be created without specifying the row and column information for each slave. This permits slaves to be rearranged, added, or removed without the need to explicitly specify row and column information. When no column or row information is specified for a *slave*, default values are chosen for column, row, columnspan and rowspan at the time the *slave* is managed. The values are chosen based upon the current layout of the grid, the position of the *slave* relative to other slaves in the same grid command, and the presence of the characters -, x, and ^ in **grid** command where *slave* names are normally expected.

**-**

This increases the columnspan of the *slave* to the left. Several '-'s in a row will successively increase the columnspan. A '-' may not follow a ^ or a x, nor may it be the first *slave* argument to **grid configure**.

**x**

This leaves an empty column between the *slave* on the left and the *slave* on the right.

**^**

This extends the rowspan of the *slave* above the '^'s in the grid. The number of '^'s in a row must match the number of columns spanned by the *slave* above it.

### THE GRID ALGORITHM

The grid geometry manager lays out its slaves in three steps. In the first
step, the minimum size needed to fit all of the slaves is computed, then (if propagation is turned on), a request is made of the master window to become that size. In the second step, the requested size is compared against the actual size of the master. If the sizes are different, then spaces is added to or taken away from the layout as needed. For the final step, each slave is positioned in its row(s) and column(s) based on the setting of its sticky flag.

To compute the minimum size of a layout, the grid geometry manager first looks at all slaves whose columnspan and rowspan values are one, and computes the nominal size of each row or column to be either the minsize for that row or column, or the sum of the padding plus the size of the largest slave, whichever is greater. After that the rows or columns in each uniform group adapt to each other. Then the slaves whose rowspans or columnspans are greater than one are examined. If a group of rows or columns need to be increased in size in order to accommodate these slaves, then extra space is added to each row or column in the group according to its weight. For each group whose weights are all zero, the additional space is apportioned equally.

When multiple rows or columns belong to a uniform group, the space allocated to them is always in proportion to their weights. (A weight of zero is considered to be 1.) In other words, a row or column configured with -weight 1 -uniform a will have exactly the same size as any other row or column configured with -weight 1 -uniform a. A row or column configured with -weight 2 -uniform b will be exactly twice as large as one that is configured with -weight 1 -uniform b.

More technically, each row or column in the group will have a size equal to k*weight for some constant k. The constant k is chosen so that no row or column becomes smaller than its minimum size. For example, if all rows or columns in a group have the same weight, then each row or column will have the same size as the largest row or column in the group.

For masters whose size is larger than the requested layout, the additional space is apportioned according to the row and column weights. If all of the weights are zero, the layout is placed within its
master according to the *anchor* value. For masters whose size is smaller than the requested layout, space is taken away from columns and rows according to their weights. However, once a column or row shrinks to its minsize, its weight is taken to be zero. If more space needs to be removed from a layout than would be permitted, as when all the rows or columns are at their minimum sizes, the layout is placed and clipped according to the *anchor* value.

---

**GEOMETRY PROPAGATION**

The grid geometry manager normally computes how large a master must be to just exactly meet the needs of its slaves, and it sets the requested width and height of the master to these dimensions. This causes geometry information to propagate up through a window hierarchy to a top-level window so that the entire sub-tree sizes itself to fit the needs of the leaf windows. However, the *grid propagate* command may be used to turn off propagation for one or more masters. If propagation is disabled then grid will not set the requested width and height of the master window. This may be useful if, for example, you wish for a master window to have a fixed size that you specify.

---

**RESTRICTIONS ON MASTER WINDOWS**

The master for each slave must either be the slave's parent (the default) or a descendant of the slave's parent. This restriction is necessary to guarantee that the slave can be placed over any part of its master that is visible without danger of the slave being clipped by its parent. In addition, all slaves in one call to *grid* must have the same master.

---

**STACKING ORDER**

If the master for a slave is not its parent then you must make sure that the slave is higher in the stacking order than the master. Otherwise the master will obscure the slave and it will appear as if the slave has not been managed correctly. The easiest way to make sure the slave is higher than the master is to create the master window first: the most recently created window will be highest in the stacking order.
CREDITS

The **grid** command is based on ideas taken from the GridBag geometry manager written by Doug. Stein, and the **blt_table** geometry manager, written by George Howlett.

EXAMPLES

A toplevel window containing a text widget and two scrollbars:

```tcl
# Make the widgets
toplevel .t
text .t.txt -wrap none -xscroll {.t.h set} -yscroll
scrollbar .t.v -orient vertical -command {.t.txt y
scrollbar .t.h -orient horizontal -command {.t.txt x

# Lay them out
grid .t.txt .t.v -sticky nsew
grid .t.h -sticky nsew

# Tell the text widget to take all the extra room
grid rowconfigure .t .t.txt -weight 1
grid columnconfigure .t .t.txt -weight 1
```

Three widgets of equal width, despite their different “natural” widths:

```tcl
button .b -text "Foo"
entry .e -variable foo
label .l -text "This is a fairly long piece of text"

grid .b .e .l -sticky ew
grid columnconfigure ."all" -uniform allTheSame
```
SEE ALSO

pack, place

KEYWORDS

gometry manager, location, grid, cell, propagation, size, pack
NAME
   photo - Full-color images
SYNOPSIS
DESCRIPTION
CREATING PHOTOS
   -data string
   -format format-name
   -file name
   -gamma value
   -height number
   -palette palette-spec
   -width number
IMAGE COMMAND
   imageName blank
   imageName cget option
   imageName configure ?option? ?value option value ...?
   imageName copy sourceImage ?option value(s) ...?
   -from x1 y1 x2 y2
   -to x1 y1 x2 y2
   -shrink
   -zoom x y
   -subsample x y
   -compositingrule rule
   imageName data ?option value(s) ...?
   -background color
   -format format-name
   -from x1 y1 x2 y2
   -grayscale
   imageName get x y
   imageName put data ?option value(s) ...?
   -format format-name
imageName read filename ?option value(s) ...?
-format format-name
-from x1 y1 x2 y2
-shrink
-to x y

imageName redither

imageName transparency subcommand ?arg arg ...?
 imageName transparency get x y
 imageName transparency set x y boolean

imageName write filename ?option value(s) ...?
-background color
-format format-name
-from x1 y1 x2 y2
-grayscale

NAME

photo - Full-color images

SYNOPSIS

image create photo ?name? ?options?

DESCRIPTION

A photo is an image whose pixels can display any color or be transparent. A photo image is stored internally in full color (32 bits per pixel), and is displayed using dithering if necessary. Image data for a photo image can be obtained from a file or a string, or it can be supplied from C code through a procedural interface. At present, only GIF and PPM/PGM formats are supported, but an interface exists to allow
additional image file formats to be added easily. A photo image is transparent in regions where no image data has been supplied or where it has been set transparent by the transparency set subcommand.

CREATING PHOTOS

Like all images, photos are created using the image create command. Photos support the following options:

-data string
   Specifies the contents of the image as a string. The string should contain binary data or, for some formats, base64-encoded data (this is currently guaranteed to be supported for GIF images). The format of the string must be one of those for which there is an image file format handler that will accept string data. If both the -data and -file options are specified, the -file option takes precedence.

-format format-name
   Specifies the name of the file format for the data specified with the -data or -file option.

-file name
   name gives the name of a file that is to be read to supply data for the photo image. The file format must be one of those for which there is an image file format handler that can read data.

-gamma value
   Specifies that the colors allocated for displaying this image in a window should be corrected for a non-linear display with the specified gamma exponent value. (The intensity produced by most CRT displays is a power function of the input value, to a good approximation; gamma is the exponent and is typically around 2). The value specified must be greater than zero. The default value is one (no correction). In general, values greater than one will make the image lighter, and values less than one will make it darker.

-height number
Specifies the height of the image, in pixels. This option is useful primarily in situations where the user wishes to build up the contents of the image piece by piece. A value of zero (the default) allows the image to expand or shrink vertically to fit the data stored in it.

**-palette palette-spec**

Specifies the resolution of the color cube to be allocated for displaying this image, and thus the number of colors used from the colormaps of the windows where it is displayed. The *palette-spec* string may be either a single decimal number, specifying the number of shades of gray to use, or three decimal numbers separated by slashes (/), specifying the number of shades of red, green and blue to use, respectively. If the first form (a single number) is used, the image will be displayed in monochrome (i.e., grayscale).

**-width number**

Specifies the width of the image, in pixels. This option is useful primarily in situations where the user wishes to build up the contents of the image piece by piece. A value of zero (the default) allows the image to expand or shrink horizontally to fit the data stored in it.

---

**IMAGE COMMAND**

When a photo image is created, Tk also creates a new command whose name is the same as the image. This command may be used to invoke various operations on the image. It has the following general form:

```
imageName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command.

Those options that write data to the image generally expand the size of
the image, if necessary, to accommodate the data written to the image, unless the user has specified non-zero values for the -width and/or -height configuration options, in which case the width and/or height, respectively, of the image will not be changed.

The following commands are possible for photo images:

`imageName blank`
Blank the image; that is, set the entire image to have no data, so it will be displayed as transparent, and the background of whatever window it is displayed in will show through.

`imageName cget option`
Returns the current value of the configuration option given by `option`. Option may have any of the values accepted by the `image create photo` command.

`imageName configure ?option? ?value option value ...?`
Query or modify the configuration options for the image. If no `option` is specified, returns a list describing all of the available options for `imageName` (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named `option` (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option-value` pairs are specified, then the command modifies the given `option(s)` to have the given `value(s)`; in this case the command returns an empty string. `Option` may have any of the values accepted by the `image create photo` command.

`imageName copy sourceImage ?option value(s) ...?`
Copies a region from the image called `sourceImage` (which must be a photo image) to the image called `imageName`, possibly with pixel zooming and/or subsampling. If no options are specified, this command copies the whole of `sourceImage` into `imageName`, starting at coordinates (0,0) in `imageName`. The following options may be specified:
-from \(x_1\) \(y_1\) \(x_2\) \(y_2\)
  Specifies a rectangular sub-region of the source image to be copied. \((x_1,y_1)\) and \((x_2,y_2)\) specify diagonally opposite corners of the rectangle. If \(x_2\) and \(y_2\) are not specified, the default value is the bottom-right corner of the source image. The pixels copied will include the left and top edges of the specified rectangle but not the bottom or right edges. If the -from option is not given, the default is the whole source image.

-to \(x_1\) \(y_1\) \(x_2\) \(y_2\)
  Specifies a rectangular sub-region of the destination image to be affected. \((x_1,y_1)\) and \((x_2,y_2)\) specify diagonally opposite corners of the rectangle. If \(x_2\) and \(y_2\) are not specified, the default value is \((x_1,y_1)\) plus the size of the source region (after subsampling and zooming, if specified). If \(x_2\) and \(y_2\) are specified, the source region will be replicated if necessary to fill the destination region in a tiled fashion.

-shrink
  Specifies that the size of the destination image should be reduced, if necessary, so that the region being copied into is at the bottom-right corner of the image. This option will not affect the width or height of the image if the user has specified a non-zero value for the -width or -height configuration option, respectively.

-zoom \(x\) \(y\)
  Specifies that the source region should be magnified by a factor of \(x\) in the X direction and \(y\) in the Y direction. If \(y\) is not given, the default value is the same as \(x\). With this option, each pixel in the source image will be expanded into a block of \(x\times y\) pixels in the destination image, all the same color. \(x\) and \(y\) must be greater than 0.

-subsample \(x\) \(y\)
  Specifies that the source image should be reduced in size by using only every \(x\)th pixel in the X direction and \(y\)th pixel in the Y direction. Negative values will cause the image to be flipped
about the Y or X axes, respectively. If y is not given, the default value is the same as x.

**-compositingrule** *rule*
Specifies how transparent pixels in the source image are combined with the destination image. When a compositing rule of *overlay* is set, the old contents of the destination image are visible, as if the source image were printed on a piece of transparent film and placed over the top of the destination. When a compositing rule of *set* is set, the old contents of the destination image are discarded and the source image is used as-is. The default compositing rule is *overlay*.

*imageName data ?option value(s) ...?*
Returns image data in the form of a string. The following options may be specified:

**-background** *color*
If the color is specified, the data will not contain any transparency information. In all transparent pixels the color will be replaced by the specified color.

**-format** *format-name*
Specifies the name of the image file format handler to be used. Specifically, this subcommand searches for the first handler whose name matches an initial substring of *format-name* and which has the capability to read this image data. If this option is not given, this subcommand uses the first handler that has the capability to read the image data.

**-from** *x1 y1 x2 y2*
Specifies a rectangular region of *imageName* to be returned. If only *x1* and *y1* are specified, the region extends from (*x1*,*y1*) to the bottom-right corner of *imageName*. If all four coordinates are given, they specify diagonally opposite corners of the rectangular region, including *x1*,*y1* and excluding *x2*,*y2*. The default, if this option is not given, is the whole image.
-grayscale
   If this options is specified, the data will not contain color information. All pixel data will be transformed into grayscale.

imageName get x y
Returns the color of the pixel at coordinates (x,y) in the image as a list of three integers between 0 and 255, representing the red, green and blue components respectively.

imageName put data ?option value(s) ...?
Sets pixels in imageName to the data specified in data. This command first searches the list of image file format handlers for a handler that can interpret the data in data, and then reads the image encoded within into imageName (the destination image). If data does not match any known format, an attempt to interpret it as a (top-to-bottom) list of scan-lines is made, with each scan-line being a (left-to-right) list of pixel colors (see Tk_GetColor for a description of valid colors.) Every scan-line must be of the same length. Note that when data is a single color name, you are instructing Tk to fill a rectangular region with that color. The following options may be specified:

-format format-name
   Specifies the format of the image data in data. Specifically, only image file format handlers whose names begin with format-name will be used while searching for an image data format handler to read the data.

-to x1 y1 ?x2 y2?
   Specifies the coordinates of the top-left corner (x1,y1) of the region of imageName into which data from filename are to be read. The default is (0,0). If x2,y2 is given and data is not large enough to cover the rectangle specified by this option, the image data extracted will be tiled so it covers the entire destination rectangle. Note that if data specifies a single color value, then a region extending to the bottom-right corner represented by (x2,y2) will be filled with that color.
imageName read filename ?option value(s) ...?

Reads image data from the file named filename into the image. This command first searches the list of image file format handlers for a handler that can interpret the data in filename, and then reads the image in filename into imageName (the destination image). The following options may be specified:

-format format-name
   Specifies the format of the image data in filename. Specifically, only image file format handlers whose names begin with format-name will be used while searching for an image data format handler to read the data.

-from x1 y1 x2 y2
   Specifies a rectangular sub-region of the image file data to be copied to the destination image. If only x1 and y1 are specified, the region extends from (x1,y1) to the bottom-right corner of the image in the image file. If all four coordinates are specified, they specify diagonally opposite corners or the region. The default, if this option is not specified, is the whole of the image in the image file.

-shrink
   If this option, the size of imageName will be reduced, if necessary, so that the region into which the image file data are read is at the bottom-right corner of the imageName. This option will not affect the width or height of the image if the user has specified a non-zero value for the -width or -height configuration option, respectively.

-to x y
   Specifies the coordinates of the top-left corner of the region of imageName into which data from filename are to be read. The default is (0,0).

imageName redither
   The dithering algorithm used in displaying photo images propagates quantization errors from one pixel to its neighbors. If
the image data for imageName is supplied in pieces, the dithered image may not be exactly correct. Normally the difference is not noticeable, but if it is a problem, this command can be used to recalculate the dithered image in each window where the image is displayed.

imageName transparency subcommand ?arg arg ...?  
Allows examination and manipulation of the transparency information in the photo image. Several subcommands are available:

imageName transparency get x y  
Returns a boolean indicating if the pixel at (x,y) is transparent.

imageName transparency set x y boolean  
Makes the pixel at (x,y) transparent if boolean is true, and makes that pixel opaque otherwise.

imageName write filename ?option value(s) ...?  
Writes image data from imageName to a file named filename. The following options may be specified:

-background color  
If the color is specified, the data will not contain any transparency information. In all transparent pixels the color will be replaced by the specified color.

-format format-name  
Specifies the name of the image file format handler to be used to write the data to the file. Specifically, this subcommand searches for the first handler whose name matches an initial substring of format-name and which has the capability to write an image file. If this option is not given, this subcommand uses the first handler that has the capability to write an image file.

-from x1 y1 x2 y2  
Specifies a rectangular region of imageName to be written to the image file. If only x1 and y1 are specified, the region
extends from \((x_1,y_1)\) to the bottom-right corner of \(imageName\). If all four coordinates are given, they specify diagonally opposite corners of the rectangular region. The default, if this option is not given, is the whole image.

-grayscale

If this options is specified, the data will not contain color information. All pixel data will be transformed into grayscale.

### IMAGE FORMATS

The photo image code is structured to allow handlers for additional image file formats to be added easily. The photo image code maintains a list of these handlers. Handlers are added to the list by registering them with a call to `Tk_CreatePhotoImageFormat`. The standard Tk distribution comes with handlers for PPM/PGM and GIF formats, which are automatically registered on initialization.

When reading an image file or processing string data specified with the `-data` configuration option, the photo image code invokes each handler in turn until one is found that claims to be able to read the data in the file or string. Usually this will find the correct handler, but if it does not, the user may give a format name with the `-format` option to specify which handler to use. In fact the photo image code will try those handlers whose names begin with the string specified for the `-format` option (the comparison is case-insensitive). For example, if the user specifies `-format gif`, then a handler named GIF87 or GIF89 may be invoked, but a handler named JPEG may not (assuming that such handlers had been registered).

When writing image data to a file, the processing of the `-format` option is slightly different: the string value given for the `-format` option must begin with the complete name of the requested handler, and may contain additional information following that, which the handler can use, for example, to specify which variant to use of the formats supported by the handler. Note that not all image handlers may support writing transparency data to a file, even where the target image format does.
COLOR ALLOCATION

When a photo image is displayed in a window, the photo image code allocates colors to use to display the image and dithers the image, if necessary, to display a reasonable approximation to the image using the colors that are available. The colors are allocated as a color cube, that is, the number of colors allocated is the product of the number of shades of red, green and blue.

Normally, the number of colors allocated is chosen based on the depth of the window. For example, in an 8-bit PseudoColor window, the photo image code will attempt to allocate seven shades of red, seven shades of green and four shades of blue, for a total of 198 colors. In a 1-bit StaticGray (monochrome) window, it will allocate two colors, black and white. In a 24-bit DirectColor or TrueColor window, it will allocate 256 shades each of red, green and blue. Fortunately, because of the way that pixel values can be combined in DirectColor and TrueColor windows, this only requires 256 colors to be allocated. If not all of the colors can be allocated, the photo image code reduces the number of shades of each primary color and tries again.

The user can exercise some control over the number of colors that a photo image uses with the -palette configuration option. If this option is used, it specifies the maximum number of shades of each primary color to try to allocate. It can also be used to force the image to be displayed in shades of gray, even on a color display, by giving a single number rather than three numbers separated by slashes.

CREDITS

The photo image type was designed and implemented by Paul Mackerras, based on his earlier photo widget and some suggestions from John Ousterhout.

EXAMPLE

Load an image from a file and tile it to the size of a window, which is useful for producing a tiled background:
# These lines should be called once
image create photo untiled -file "theFile.ppm"
image create photo tiled

# These lines should be called whenever .someWidget
# size; a <Configure> binding is useful here
set width [winfo width .someWidget]
set height [winfo height .someWidget]
tiled copy untiled -to 0 0 $width $height -shrink

SEE ALSO

image

KEYWORDS

photo, image, color

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Australian National University
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME

tk_focusNext, tk_focusPrev, tk_focusFollowsMouse - Utility procedures for managing the input focus.

SYNOPSIS

```
tk_focusNext window
tk_focusPrev window
tk_focusFollowsMouse
```

DESCRIPTION

**tk_focusNext** is a utility procedure used for keyboard traversal. It returns the “next” window after *window* in focus order. The focus order is determined by the stacking order of windows and the structure of the window hierarchy. Among siblings, the focus order is the same as the stacking order, with the lowest window being first. If a window has children, the window is visited first, followed by its children (recursively), followed by its next sibling. Top-level windows other than *window* are skipped, so that **tk_focusNext** never returns a window in a different top-level from *window*.

After computing the next window, **tk_focusNext** examines the window's `-takefocus` option to see whether it should be skipped. If so, **tk_focusNext** continues on to the next window in the focus order, until it eventually finds a window that will accept the focus or returns back to *window*.

**tk_focusPrev** is similar to **tk_focusNext** except that it returns the window just before *window* in the focus order.
tk_focusFollowsMouse changes the focus model for the application to an implicit one where the window under the mouse gets the focus. After this procedure is called, whenever the mouse enters a window Tk will automatically give it the input focus. The focus command may be used to move the focus to a window other than the one under the mouse, but as soon as the mouse moves into a new window the focus will jump to that window. Note: at present there is no built-in support for returning the application to an explicit focus model; to do this you will have to write a script that deletes the bindings created by tk_focusFollowsMouse.

**KEYWORDS**

focus, keyboard traversal, top-level

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

tkwait - Wait for variable to change or window to be destroyed

SYNOPSIS

```
tkwait variable name
tkwait visibility name
tkwait window name
```

DESCRIPTION

The `tkwait` command waits for one of several things to happen, then it returns without taking any other actions. The return value is always an empty string. If the first argument is `variable` (or any abbreviation of it) then the second argument is the name of a global variable and the command waits for that variable to be modified. If the first argument is `visibility` (or any abbreviation of it) then the second argument is the name of a window and the `tkwait` command waits for a change in its visibility state (as indicated by the arrival of a VisibilityNotify event). This form is typically used to wait for a newly-created window to appear on the screen before taking some action. If the first argument is `window` (or any abbreviation of it) then the second argument is the name of a window and the `tkwait` command waits for that window to be destroyed. This form is typically used to wait for a user to finish interacting with a dialog box before using the result of that interaction.

While the `tkwait` command is waiting it processes events in the normal fashion, so the application will continue to respond to user interactions. If an event handler invokes `tkwait` again, the nested call to `tkwait` must complete before the outer call can complete.
variable, visibility, wait, window

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1992 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
  ttk::scale - Create and manipulate a scale widget
SYNOPSIS
DESCRIPTION
STANDARD OPTIONS
  -class
  -cursor, cursor, Cursor
  -style
  -takefocus, takeFocus, TakeFocus
WIDGET-SPECIFIC OPTIONS
  -command, command, Command
  -from, from, From
  -length, length, Length
  -orient, orient, Orient
  -to, to, To
  -value, value, Value
  -variable, variable, Variable
WIDGET COMMAND
  pathName cget option
  pathName configure ?option? ?value option value ...?
  pathName get ?x y?
  pathName identify x y
  pathName instate statespec ?script?
  pathName set value
  pathName state ?stateSpec?
INTERNAL COMMANDS
  pathName coords ?value?
SEE ALSO
KEYWORDS
NAME
ttk::scale - Create and manipulate a scale widget

SYNOPSIS

ttk::scale pathName ?options...?

DESCRIPTION

A ttk::scale widget is typically used to control the numeric value of a linked variable that varies uniformly over some range. A scale displays a slider that can be moved along over a trough, with the relative position of the slider over the trough indicating the value of the variable.

STANDARD OPTIONS

-class
-cursor, cursor, Cursor
-style
-takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS

Command-Line Name: -command
Database Name: command
Database Class: Command
    Specifies the prefix of a Tcl command to invoke whenever the scale's value is changed via a widget command. The actual command consists of this option followed by a space and a real number indicating the new value of the scale.

Command-Line Name: -from
Database Name: from
Database Class: From
    A real value corresponding to the left or top end of the scale.

Command-Line Name: -length
Database Name: length
Database Class: Length
Specifies the desired long dimension of the scale in screen units (i.e. any of the forms acceptable to Tk_GetPixels). For vertical scales this is the scale's height; for horizontal scales it is the scale's width.

**Command-Line Name: -orient**
**Database Name: orient**
**Database Class: Orient**
Specifies which orientation whether the widget should be laid out horizontally or vertically. Must be either **horizontal** or **vertical** or an abbreviation of one of these.

**Command-Line Name: -to**
**Database Name: to**
**Database Class: To**
Specifies a real value corresponding to the right or bottom end of the scale. This value may be either less than or greater than the **from** option.

**Command-Line Name: -value**
**Database Name: value**
**Database Class: Value**
Specifies the current floating-point value of the variable.

**Command-Line Name: -variable**
**Database Name: variable**
**Database Class: Variable**
Specifies the name of a global variable to link to the scale. Whenever the value of the variable changes, the scale will update to reflect this value. Whenever the scale is manipulated interactively, the variable will be modified to reflect the scale's new value.

**WIDGET COMMAND**

`pathName cget option`
Returns the current value of the specified **option**; see *ttk::widget(n)*.
**pathName configure option? value option value ...?**
Modify or query widget options; see ttk::widget(n).

**pathName get x y?**
Get the current value of the -value option, or the value corresponding to the coordinates x,y if they are specified. X and y are pixel coordinates relative to the scale widget origin.

**pathName identify x y**
Returns the name of the element at position x, y. See ttk::widget(n).

**pathName instate statespec ?script?**
Test the widget state; see ttk::widget(n).

**pathName set value**
Set the value of the widget (i.e. the -value option) to value. The value will be clipped to the range given by the -from and -to options. Note that setting the linked variable (i.e. the variable named in the -variable option) does not cause such clipping.

**pathName state stateSpec ?**
Modify or query the widget state; see ttk::widget(n).

---

**INTERNAL COMMANDS**

**pathName coords value?**
Get the coordinates corresponding to value, or the coordinates corresponding to the current value of the -value option if value is omitted.

---

**SEE ALSO**

ttk::widget, scale

---

**KEYWORDS**

scale, slider, trough, widget

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
button - Create and manipulate button widgets

SYNOPSIS
STANDARD OPTIONS
-activebackground, activeBackground, Foreground
-foregroundforeground, activeForeground, Background
-foregroundanchor, anchor, Anchor
-foreground background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foregroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergroundergri


WIDGET-SPECIFIC OPTIONS
- command, command, Command
- default, default, Default
- height, height, Height
- overrelief, overRelief, OverRelief
- state, state, State
- width, width, Width

DESCRIPTION
WIDGET COMMAND
pathName cget option
pathName configure ?option? ?value option value ...?
pathName flash
pathName invoke

DEFAULT BINDINGS

EXAMPLES

SEE ALSO

KEYWORDS

NAME

button - Create and manipulate button widgets

SYNOPSIS

button pathName ?options?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-repeatdelay, repeatDelay, RepeatDelay
-repeatinterval, repeatInterval, RepeatInterval
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

WIDGET-SPECIFIC OPTIONS

Command-Line Name: -command
Database Name: command
Database Class: Command
  Specifies a Tcl command to associate with the button. This command is typically invoked when mouse button 1 is released over the button window.

Command-Line Name: -default
Database Name: default
Database Class: Default
  Specifies one of three states for the default ring: normal, active, or disabled. In active state, the button is drawn with the platform specific appearance for a default button. In normal state, the button is drawn with the platform specific appearance for a non-default button, leaving enough space to draw the default button appearance. The normal and active states will result in buttons of the same size. In disabled state, the button is drawn with the non-default button appearance without leaving space for the default
appearance. The disabled state may result in a smaller button than the active state.

Command-Line Name: -height
Database Name: height
Database Class: Height
Specifies a desired height for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to `Tk_GetPixels`); for text it is in lines of text. If this option is not specified, the button's desired height is computed from the size of the image or bitmap or text being displayed in it.

Command-Line Name: -overrelief
Database Name: overRelief
Database Class: OverRelief
Specifies an alternative relief for the button, to be used when the mouse cursor is over the widget. This option can be used to make toolbar buttons, by configuring `-relief flat -overrelief raised`. If the value of this option is the empty string, then no alternative relief is used when the mouse cursor is over the button. The empty string is the default value.

Command-Line Name: -state
Database Name: state
Database Class: State
Specifies one of three states for the button: normal, active, or disabled. In normal state the button is displayed using the foreground and background options. The active state is typically used when the pointer is over the button. In active state the button is displayed using the activeForeground and activeBackground options. Disabled state means that the button should be insensitive: the default bindings will refuse to activate the widget and will ignore mouse button presses. In this state the disabledForeground and background options determine how the button is displayed.

Command-Line Name: -width
Database Name: **width**
Database Class: **Width**

Specifies a desired width for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to **Tk_GetPixels**). For a text button (no image or with **-compound none**) then the width specifies how much space in characters to allocate for the text label. If the width is negative then this specifies a minimum width. If this option is not specified, the button's desired width is computed from the size of the image or bitmap or text being displayed in it.

**DESCRIPTION**

The **button** command creates a new window (given by the **pathName** argument) and makes it into a button widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the button such as its colors, font, text, and initial relief. The **button** command returns its **pathName** argument. At the time this command is invoked, there must not exist a window named **pathName**, but **pathName**'s parent must exist.

A button is a widget that displays a textual string, bitmap or image. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the **wrapLength** option) and one of the characters may optionally be underlined using the **underline** option. It can display itself in either of three different ways, according to the **state** option; it can be made to appear raised, sunken, or flat; and it can be made to flash. When a user invokes the button (by pressing mouse button 1 with the cursor over the button), then the Tcl command specified in the **-command** option is invoked.

**WIDGET COMMAND**

The **button** command creates a new Tcl command whose name is **pathName**. This command may be used to invoke various operations on the widget. It has the following general form:
Option and the args determine the exact behavior of the command. The following commands are possible for button widgets:

**pathName cget option**
Returns the current value of the configuration option given by option. Option may have any of the values accepted by the button command.

**pathName configure ?option? ?value option value ...?**
Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option-value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the button command.

**pathName flash**
Flash the button. This is accomplished by redisplaying the button several times, alternating between active and normal colors. At the end of the flash the button is left in the same normal/active state as when the command was invoked. This command is ignored if the button's state is disabled.

**pathName invoke**
Invoke the Tcl command associated with the button, if there is one. The return value is the return value from the Tcl command, or an empty string if there is no command associated with the button. This command is ignored if the button's state is disabled.
Tk automatically creates class bindings for buttons that give them default behavior:

[1] A button activates whenever the mouse passes over it and deactivates whenever the mouse leaves the button. Under Windows, this binding is only active when mouse button 1 has been pressed over the button.

[2] A button's relief is changed to sunken whenever mouse button 1 is pressed over the button, and the relief is restored to its original value when button 1 is later released.

[3] If mouse button 1 is pressed over a button and later released over the button, the button is invoked. However, if the mouse is not over the button when button 1 is released, then no invocation occurs.

[4] When a button has the input focus, the space key causes the button to be invoked.

If the button's state is **disabled** then none of the above actions occur: the button is completely non-responsive.

The behavior of buttons can be changed by defining new bindings for individual widgets or by redefining the class bindings.

### EXAMPLES

This is the classic Tk “Hello, World!” demonstration:

```
button .b -text "Hello, World!" -command exit
pack .b
```
This example demonstrates how to handle button accelerators:

```python
button .b1 -text Hello -underline 0
button .b2 -text World -underline 0
bind . <Key-h> {.b1 flash; .b1 invoke}
bind . <Key-w> {.b2 flash; .b2 invoke}
pack .b1 .b2
```

SEE ALSO

```python
ttk::button
```

KEYWORDS

```
button, widget
```

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

image - Create and manipulate images

SYNOPSIS

DESCRIPTION

image create type ?name? ?option value ...?
image delete ?name name ...?
image height name
image inuse name
image names
image type name
image types
image width name

BUILT-IN IMAGE TYPES

bitmap
photo

SEE ALSO

KEYWORDS

NAME

image - Create and manipulate images

SYNOPSIS

image option ?arg arg ...?

DESCRIPTION

The image command is used to create, delete, and query images. It can take several different forms, depending on the option argument. The legal forms are:
image create type ?name? ?option value ...?
Creates a new image and a command with the same name and returns its name. type specifies the type of the image, which must be one of the types currently defined (e.g., bitmap). name specifies the name for the image; if it is omitted then Tk picks a name of the form imagex, where x is an integer. There may be any number of option-value pairs, which provide configuration options for the new image. The legal set of options is defined separately for each image type; see below for details on the options for built-in image types. If an image already exists by the given name then it is replaced with the new image and any instances of that image will redisplay with the new contents. It is important to note that the image command will silently overwrite any procedure that may currently be defined by the given name, so choose the name wisely. It is recommended to use a separate namespace for image names (e.g., ::img::logo, ::img::large).

image delete ?name name ...?
Deletes each of the named images and returns an empty string. If there are instances of the images displayed in widgets, the images will not actually be deleted until all of the instances are released. However, the association between the instances and the image manager will be dropped. Existing instances will retain their sizes but redisplay as empty areas. If a deleted image is recreated with another call to image create, the existing instances will use the new image.

image height name
Returns a decimal string giving the height of image name in pixels.

image inuse name
Returns a boolean value indicating whether or not the image given by name is in use by any widgets.

image names
Returns a list containing the names of all existing images.

image type name
Returns the type of image name (the value of the type argument to `image create` when the image was created).

**image types**

Returns a list whose elements are all of the valid image types (i.e., all of the values that may be supplied for the type argument to `image create`).

**image width** name

Returns a decimal string giving the width of image name in pixels.

Additional operations (e.g. writing the image to a file) may be available as subcommands of the image instance command. See the manual page for the particular image type for details.

### BUILT-IN IMAGE TYPES

The following image types are defined by Tk so they will be available in any Tk application. Individual applications or extensions may define additional types.

**bitmap**

Each pixel in the image displays a foreground color, a background color, or nothing. See the `bitmap` manual entry for more information.

**photo**

Displays a variety of full-color images, using dithering to approximate colors on displays with limited color capabilities. See the `photo` manual entry for more information.

### SEE ALSO

`bitmap`, `options`, `photo`

### KEYWORDS

`height`, `image`, `types of images`, `width`
NAME

place - Geometry manager for fixed or rubber-sheet placement

SYNOPSIS

DESCRIPTION

place window option value ?option value ...?
place configure window ?option? ?value option value ...?

-anchor where
-bordermode mode
-height size
-in master
-relheight size
-relwidth size
-relx location
-rely location
-width size
-x location
-y location

place forget window
place info window
place slaves window

FINE POINTS
EXAMPLE
SEE ALSO
KEYWORDS

NAME

place - Geometry manager for fixed or rubber-sheet placement

SYNOPSIS

place option arg ?arg ...?
DESCRIPTION

The placer is a geometry manager for Tk. It provides simple fixed placement of windows, where you specify the exact size and location of one window, called the *slave*, within another window, called the *master*. The placer also provides rubber-sheet placement, where you specify the size and location of the slave in terms of the dimensions of the master, so that the slave changes size and location in response to changes in the size of the master. Lastly, the placer allows you to mix these styles of placement so that, for example, the slave has a fixed width and height but is centered inside the master.

place *window* option value ?option value ...?
Arrange for the placer to manage the geometry of a slave whose pathName is *window*. The remaining arguments consist of one or more *option-value* pairs that specify the way in which *window*’s geometry is managed. *Option* may have any of the values accepted by the place configure command.

place configure *window* ?option? ?value option value ...?  
Query or modify the geometry options of the slave given by *window*. If no *option* is specified, this command returns a list describing the available options (see Tk_ConfigureInfo for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option-value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. The following *option-value* pairs are supported:

-anchor *where*
  *Where* specifies which point of *window* is to be positioned at the (x,y) location selected by the -x, -y, -relx, and -rely options. The anchor point is in terms of the outer area of *window* including its border, if any. Thus if *where* is se then the lower-right corner of *window*’s border will appear at the given (x,y) location in the master. The anchor position defaults to nw.
-bordermode *mode*

*Mode* determines the degree to which borders within the master are used in determining the placement of the slave. The default and most common value is **inside**. In this case the placer considers the area of the master to be the innermost area of the master, inside any border: an option of **-x 0** corresponds to an x-coordinate just inside the border and an option of **-relwidth 1.0** means *window* will fill the area inside the master's border. If *mode* is **outside** then the placer considers the area of the master to include its border; this mode is typically used when placing *window* outside its master, as with the options **-x 0 -y 0 -anchor ne**. Lastly, *mode* may be specified as **ignore**, in which case borders are ignored: the area of the master is considered to be its official X area, which includes any internal border but no external border. A bordermode of **ignore** is probably not very useful.

-height *size*

*Size* specifies the height for *window* in screen units (i.e. any of the forms accepted by **Tk_GetPixels**). The height will be the outer dimension of *window* including its border, if any. If *size* is an empty string, or if no **-height** or **-relheight** option is specified, then the height requested internally by the window will be used.

-in *master*

*Master* specifies the path name of the window relative to which *window* is to be placed. *Master* must either be *window*'s parent or a descendant of *window*'s parent. In addition, *master* and *window* must both be descendants of the same top-level window. These restrictions are necessary to guarantee that *window* is visible whenever *master* is visible. If this option is not specified then the master defaults to *window*'s parent.

-relheight *size*

*Size* specifies the height for *window*. In this case the height is specified as a floating-point number relative to the height of the master: 0.5 means *window* will be half as high as the master,
1.0 means *window* will have the same height as the master, and so on. If both `-height` and `-relheight` are specified for a slave, their values are summed. For example, `-relheight 1.0 -height -2` makes the slave 2 pixels shorter than the master.

**-relwidth size**

*Size* specifies the width for *window*. In this case the width is specified as a floating-point number relative to the width of the master: 0.5 means *window* will be half as wide as the master, 1.0 means *window* will have the same width as the master, and so on. If both `-width` and `-relwidth` are specified for a slave, their values are summed. For example, `-relwidth 1.0 -width 5` makes the slave 5 pixels wider than the master.

**-relx location**

*Location* specifies the x-coordinate within the master window of the anchor point for *window*. In this case the location is specified in a relative fashion as a floating-point number: 0.0 corresponds to the left edge of the master and 1.0 corresponds to the right edge of the master. *Location* need not be in the range 0.0-1.0. If both `-x` and `-relx` are specified for a slave then their values are summed. For example, `-relx 0.5 -x -2` positions the left edge of the slave 2 pixels to the left of the center of its master.

**-rely location**

*Location* specifies the y-coordinate within the master window of the anchor point for *window*. In this case the value is specified in a relative fashion as a floating-point number: 0.0 corresponds to the top edge of the master and 1.0 corresponds to the bottom edge of the master. *Location* need not be in the range 0.0-1.0. If both `-y` and `-rely` are specified for a slave then their values are summed. For example, `-rely 0.5 -x 3` positions the top edge of the slave 3 pixels below the center of its master.

**-width size**

*Size* specifies the width for *window* in screen units (i.e. any of
the forms accepted by \texttt{Tk\_GetPixels}). The width will be the outer width of \texttt{window} including its border, if any. If \texttt{size} is an empty string, or if no \texttt{-width} or \texttt{-relwidth} option is specified, then the width requested internally by the window will be used.

\textbf{-x location}

\textit{Location} specifies the x-coordinate within the master window of the anchor point for \texttt{window}. The location is specified in screen units (i.e. any of the forms accepted by \texttt{Tk\_GetPixels}) and need not lie within the bounds of the master window.

\textbf{-y location}

\textit{Location} specifies the y-coordinate within the master window of the anchor point for \texttt{window}. The location is specified in screen units (i.e. any of the forms accepted by \texttt{Tk\_GetPixels}) and need not lie within the bounds of the master window.

If the same value is specified separately with two different options, such as \texttt{-x} and \texttt{-relx}, then the most recent option is used and the older one is ignored.

\textbf{place forget \texttt{window}}

Causes the placer to stop managing the geometry of \texttt{window}. As a side effect of this command \texttt{window} will be unmapped so that it does not appear on the screen. If \texttt{window} is not currently managed by the placer then the command has no effect. This command returns an empty string.

\textbf{place info \texttt{window}}

Returns a list giving the current configuration of \texttt{window}. The list consists of \texttt{option-value} pairs in exactly the same form as might be specified to the \texttt{place configure} command.

\textbf{place slaves \texttt{window}}

Returns a list of all the slave windows for which \texttt{window} is the master. If there are no slaves for \texttt{window} then an empty string is returned.
If the configuration of a window has been retrieved with place info, that configuration can be restored later by first using place forget to erase any existing information for the window and then invoking place configure with the saved information.

**FINE POINTS**

It is not necessary for the master window to be the parent of the slave window. This feature is useful in at least two situations. First, for complex window layouts it means you can create a hierarchy of subwindows whose only purpose is to assist in the layout of the parent. The “real children” of the parent (i.e. the windows that are significant for the application's user interface) can be children of the parent yet be placed inside the windows of the geometry-management hierarchy. This means that the path names of the “real children” do not reflect the geometry-management hierarchy and users can specify options for the real children without being aware of the structure of the geometry-management hierarchy.

A second reason for having a master different than the slave's parent is to tie two siblings together. For example, the placer can be used to force a window always to be positioned centered just below one of its siblings by specifying the configuration

```
-in sibling -relx 0.5 -rely 1.0 -anchor n -bordermode
```

Whenever the sibling is repositioned in the future, the slave will be repositioned as well.

Unlike many other geometry managers (such as the packer) the placer does not make any attempt to manipulate the geometry of the master windows or the parents of slave windows (i.e. it does not set their requested sizes). To control the sizes of these windows, make them windows like frames and canvases that provide configuration options for this purpose.
EXAMPLE

Make the label occupy the middle bit of the toplevel, no matter how it is resized:

```
label .l -text "In the\nMiddle!" -bg black -fg white
place .l -relwidth .3 -relx .35 -relheight .3 -rely
```

SEE ALSO

`grid`, `pack`

KEYWORDS

`geometry manager`, `height`, `location`, `master`, `place`, `rubber sheet`, `slave`, `width`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1992 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
toplevel - Create and manipulate toplevel widgets

SYNOPSIS

STANDARD OPTIONS
- -borderwidth or -bd, borderWidth, BorderWidth
- -cursor, cursor, Cursor
- -highlightbackground, highlightBackground,
  HighlightBackground
- -highlightcolor, highlightColor, HighlightColor
- -highlightthickness, highlightThickness, HighlightThickness
- -padX, padX, Pad
- -padY, padY, Pad
- -relief, relief, Relief
- -takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS
- -background, background, Background
- -class, class, Class
- -colormap, colormap, Colormap
- -container, container, Container
- -height, height, Height
- -menu, menu, Menu
- -screen, 
- -use, use, Use
- -visual, visual, Visual
- -width, width, Width

DESCRIPTION

WIDGET COMMAND

pathName cget option
pathName configure ?option? ?value option value ...?

BINDINGS

SEE ALSO
KEYWORDS

NAME
toplevel - Create and manipulate toplevel widgets

SYNOPSIS
toplevel pathName ?options?

STANDARD OPTIONS
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS

Command-Line Name: -background
Database Name: background
Database Class: Background
This option is the same as the standard background option except that its value may also be specified as an empty string. In this case, the widget will display no background or border, and no colors will be consumed from its colormap for its background and border.

Command-Line Name: -class
Database Name: class
Database Class: Class
Specifies a class for the window. This class will be used when querying the option database for the window's other options, and it will also be used later for other purposes such as bindings. The
**class** option may not be changed with the **configure** widget command.

Command-Line Name: **-colormap**  
Database Name: **colormap**  
Database Class: **Colormap**  
Specifies a colormap to use for the window. The value may be either **new**, in which case a new colormap is created for the window and its children, or the name of another window (which must be on the same screen and have the same visual as **pathName**), in which case the new window will use the colormap from the specified window. If the **colormap** option is not specified, the new window uses the default colormap of its screen. This option may not be changed with the **configure** widget command.

Command-Line Name: **-container**  
Database Name: **container**  
Database Class: **Container**  
The value must be a boolean. If true, it means that this window will be used as a container in which some other application will be embedded (for example, a Tk toplevel can be embedded using the **-use** option). The window will support the appropriate window manager protocols for things like geometry requests. The window should not have any children of its own in this application. This option may not be changed with the **configure** widget command.

Command-Line Name: **-height**  
Database Name: **height**  
Database Class: **Height**  
Specifies the desired height for the window in any of the forms acceptable to **Tk_GetPixels**. If this option is less than or equal to zero then the window will not request any size at all.

Command-Line Name: **-menu**  
Database Name: **menu**  
Database Class: **Menu**  
Specifies a menu widget to be used as a menubar. On the Macintosh, the menubar will be displayed across the top of the
main monitor. On Microsoft Windows and all UNIX platforms, the menu will appear across the toplevel window as part of the window dressing maintained by the window manager.

Command-Line Name: **-screen**
Database Name:
Database Class:
  Specifies the screen on which to place the new window. Any valid screen name may be used, even one associated with a different display. Defaults to the same screen as its parent. This option is special in that it may not be specified via the option database, and it may not be modified with the **configure** widget command.

Command-Line Name: **-use**
Database Name: **use**
Database Class: **Use**
  This option is used for embedding. If the value is not an empty string, it must be the window identifier of a container window, specified as a hexadecimal string like the ones returned by the **winfo id** command. The toplevel widget will be created as a child of the given container instead of the root window for the screen. If the container window is in a Tk application, it must be a frame or toplevel widget for which the **-container** option was specified. This option may not be changed with the **configure** widget command.

Command-Line Name: **-visual**
Database Name: **visual**
Database Class: **Visual**
  Specifies visual information for the new window in any of the forms accepted by **Tk_GetVisual**. If this option is not specified, the new window will use the default visual for its screen. The **visual** option may not be modified with the **configure** widget command.

Command-Line Name: **-width**
Database Name: **width**
Database Class: **Width**
  Specifies the desired width for the window in any of the forms acceptable to **Tk_GetPixels**. If this option is less than or equal to
zero then the window will not request any size at all.

**DESCRIPTION**

The `toplevel` command creates a new toplevel widget (given by the `pathName` argument). Additional options, described above, may be specified on the command line or in the option database to configure aspects of the toplevel such as its background color and relief. The `toplevel` command returns the path name of the new window.

A toplevel is similar to a frame except that it is created as a top-level window: its X parent is the root window of a screen rather than the logical parent from its path name. The primary purpose of a toplevel is to serve as a container for dialog boxes and other collections of widgets. The only visible features of a toplevel are its background color and an optional 3-D border to make the toplevel appear raised or sunken.

**WIDGET COMMAND**

The `toplevel` command creates a new Tcl command whose name is the same as the path name of the toplevel's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
    pathName option ?arg arg ...?
```

*PathName* is the name of the command, which is the same as the toplevel widget's path name. *Option* and the *args* determine the exact behavior of the command. The following commands are possible for toplevel widgets:

*pathName cget option*

- Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the `toplevel` command.
**pathName configure** ?option? ?value option value ...?

Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see `Tk_ConfigureInfo` for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option-value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the `toplevel` command.

### BINDINGS

When a new toplevel is created, it has no default event bindings: toplevels are not intended to be interactive.

### SEE ALSO

- `frame`

### KEYWORDS

- `toplevel`, `widget`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

ttk::scrollbar - Control the viewport of a scrollable widget

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS

-class
-cursor, cursor, Cursor
-style
-takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS

-command, command, Command
-orient, orient, Orient

WIDGET COMMAND

pathName cget option
pathName configure ?option? ?value option value ...?
pathName get
pathName identify x y
pathName instate statespec ?script?
pathName set first last
pathName state ?stateSpec?

INTERNAL COMMANDS

pathName delta deltaX deltaY
pathName fraction x y

SCROLLING COMMANDS

prefix moveto fraction
prefix scroll number units
prefix scroll number pages

WIDGET STATES

EXAMPLE

SEE ALSO

KEYWORDS
NAME

ttk::scrollbar - Control the viewport of a scrollable widget

SYNOPSIS

**ttk::scrollbar** *pathName* ?*options*...?

DESCRIPTION

**ttk::scrollbar** widgets are typically linked to an associated window that displays a document of some sort, such as a file being edited or a drawing. A scrollbar displays a *thumb* in the middle portion of the scrollbar, whose position and size provides information about the portion of the document visible in the associated window. The thumb may be dragged by the user to control the visible region. Depending on the theme, two or more arrow buttons may also be present; these are used to scroll the visible region in discrete units.

STANDARD OPTIONS

- **-class**
- **-cursor, cursor, Cursor**
- **-style**
- **-takefocus, takeFocus, TakeFocus**

WIDGET-SPECIFIC OPTIONS

Command-Line Name: **-command**
Database Name: **command**
Database Class: **Command**

A Tcl script prefix to evaluate to change the view in the widget associated with the scrollbar. Additional arguments are appended to the value of this option, as described in **SCROLLING COMMANDS** below, whenever the user requests a view change by manipulating the scrollbar.

This option typically consists of a two-element list, containing the
name of a scrollable widget followed by either \texttt{xview} (for horizontal scrollbars) or \texttt{yview} (for vertical scrollbars).

Command-Line Name: \texttt{-orient}
Database Name: \texttt{orient}
Database Class: \texttt{Orient}

One of \texttt{horizontal} or \texttt{vertical}. Specifies the orientation of the scrollbar.

\section*{WIDGET COMMANDS}

\texttt{pathName cget option}
Returns the current value of the specified \texttt{option}; see \texttt{ttk::widget(n)}.

\texttt{pathName configure ?option? ?value option value ...?}
Modify or query widget options; see \texttt{ttk::widget(n)}.

\texttt{pathName get}
Returns the scrollbar settings in the form of a list whose elements are the arguments to the most recent \texttt{set} widget command.

\texttt{pathName identify x y}
Returns the name of the element at position \texttt{x}, \texttt{y}. See \texttt{ttk::widget(n)}.

\texttt{pathName instate statespec ?script?}
Test the widget state; see \texttt{ttk::widget(n)}.

\texttt{pathName set first last}
This command is normally invoked by the scrollbar's associated widget from an \texttt{-xscrollcommand} or \texttt{-yscrollcommand} callback. Specifies the visible range to be displayed. \texttt{first} and \texttt{last} are real fractions between 0 and 1.

\texttt{pathName state ?stateSpec?}
Modify or query the widget state; see \texttt{ttk::widget(n)}.

\section*{INTERNAL COMMANDS}
The following widget commands are used internally by the TScrollbar
widget class bindings.

`pathName delta deltaX deltaY`
Returns a real number indicating the fractional change in the scrollbar setting that corresponds to a given change in thumb position. For example, if the scrollbar is horizontal, the result indicates how much the scrollbar setting must change to move the thumb `deltaX` pixels to the right (`deltaY` is ignored in this case). If the scrollbar is vertical, the result indicates how much the scrollbar setting must change to move the thumb `deltaY` pixels down. The arguments and the result may be zero or negative.

`pathName fraction x y`
Returns a real number between 0 and 1 indicating where the point given by `x` and `y` lies in the trough area of the scrollbar, where 0.0 corresponds to the top or left of the trough and 1.0 corresponds to the bottom or right. `X` and `y` are pixel coordinates relative to the scrollbar widget. If `x` and `y` refer to a point outside the trough, the closest point in the trough is used.

### SCROLLING COMMANDS

When the user interacts with the scrollbar, for example by dragging the thumb, the scrollbar notifies the associated widget that it must change its view. The scrollbar makes the notification by evaluating a Tcl command generated from the scrollbar's `-command` option. The command may take any of the following forms. In each case, `prefix` is the contents of the `-command` option, which usually has a form like `.t yview

`prefix moveto fraction`

*Fraction* is a real number between 0 and 1. The widget should adjust its view so that the point given by `fraction` appears at the beginning of the widget. If `fraction` is 0 it refers to the beginning of the document. 1.0 refers to the end of the document, 0.333 refers to a point one-third of the way through the document, and so on.

`prefix scroll number units`
The widget should adjust its view by \textit{number} units. The units are defined in whatever way makes sense for the widget, such as characters or lines in a text widget. \textit{Number} is either 1, which means one unit should scroll off the top or left of the window, or -1, which means that one unit should scroll off the bottom or right of the window.

\textit{prefix scroll number pages}

The widget should adjust its view by \textit{number} pages. It is up to the widget to define the meaning of a page; typically it is slightly less than what fits in the window, so that there is a slight overlap between the old and new views. \textit{Number} is either 1, which means the next page should become visible, or -1, which means that the previous page should become visible.

\section*{WIDGET STATES}

The scrollbar automatically sets the \texttt{disabled} state bit when the entire range is visible (range is 0.0 to 1.0), and clears it otherwise. It also sets the \texttt{active} and \texttt{pressed} state flags of individual elements, based on the position and state of the mouse pointer.

\section*{EXAMPLE}

\begin{verbatim}
set f [frame .f]
ttk::scrollbar $f.hsb -orient horizontal -command [list $f.t xview]
ttk::scrollbar $f.vsb -orient vertical -command [list $f.t yview]
text $f.t -xscrollcommand [list $f.hsb set] -yscrollcommand [list $f.vsb set]
grid $f.t -row 0 -column 0 -sticky nsew
grid $f.vsb -row 0 -column 1 -sticky nsew
grid $f.hsb -row 1 -column 0 -sticky nsew
grid columnconfigure $f 0 -weight 1
grid rowconfigure $f 0 -weight 1
\end{verbatim}
ttk::widget, scrollbar

KEYWORDS

scrollbar, widget
NAME
canvas - Create and manipulate canvas widgets

SYNOPSIS

STANDARD OPTIONS
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-insertbackground, insertBackground, Foreground
-insertborderwidth, insertBorderWidth, BorderWidth
-insertofftime, insertOffTime, OffTime
-insertontime, insertOnTime, OnTime
-insertwidth, insertWidth, InsertWidth
-relief, relief, Relief
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-takefocus, takeFocus, TakeFocus
-xscrollcommand, xScrollCommand, ScrollCommand
-yscrollcommand, yScrollCommand, ScrollCommand

WIDGET-SPECIFIC OPTIONS
-closeenough, closeEnough, CloseEnough
-confine, confine, Confine
-height, height, Height
-scrollregion, scrollRegion, ScrollRegion
-state, state, State
-width, width, width
-xscrollincrement, xScrollIncrement, ScrollIncrement
yscrollincrement, yScrollIncrement, ScrollIncrement

INTRODUCTION
DISPLAY LIST
ITEM IDS AND TAGS
COORDINATES
TRANSFORMATIONS
INDICES

    number
    end
    insert
    sel.first
    sel.last
    @x,y

DASH PATTERNS
WIDGET COMMAND

    pathName addtag tag searchSpec ?arg arg ...?
    above tagOrId
    all
    below tagOrId
    closest x y ?halo? ?start?
    enclosed x1 y1 x2 y2
    overlapping x1 y1 x2 y2
    withtag tagOrId

    pathName bbox tagOrId ?tagOrId tagOrId ...?
    pathName bind tagOrId ?sequence? ?command?
    pathName canvasx screenx ?gridspacing?
    pathName canvasy screeny ?gridspacing?
    pathName cget option
    pathName configure ?option? ?value? ?option value ...?
    pathName coords tagOrId ?x0 y0 ...
    pathName coords tagOrId ?coordList?
    pathName create type x y ?x y ...? ?option value ...?
    pathName create type coordList ?option value ...?
    pathName dchars tagOrId first ?last?
    pathName delete ?tagOrId tagOrId ...?
    pathName dtag tagOrId ?tagToDelete?
    pathName find searchCommand ?arg arg ...?
pathName focus ?tagOrId?
pathName gettags tagOrId
pathName icursor tagOrId index
pathName index tagOrId index
pathName insert tagOrId beforeThis string
pathName itemcget tagOrId option
pathName itemconfigure tagOrId ?option? ?value? ?option value ...?
pathName lower tagOrId ?belowThis?
pathName move tagOrId xAmount yAmount
pathName postscript ?option value option value ...?
 -colormap varName
 -colormode mode
 -file fileName
 -fontmap varName
 -height size
 -pageanchor anchor
 -pageheight size
 -pagewidth size
 -pagex position
 -pagey position
 -rotate boolean
 -width size
 -x position
 -y position
pathName raise tagOrId ?aboveThis?
pathName scale tagOrId xOrigin yOrigin xScale yScale
pathName scan option args
 pathName scan mark x y
 pathName scan dragto x y ?gain?.
pathName select option ?tagOrId arg?
 pathName select adjust tagOrId index
 pathName select clear
 pathName select from tagOrId index
 pathName select item
 pathName select to tagOrId index
pathName type tagOrId
pathName xview ?args?
    pathName xview
    pathName xview moveto fraction
    pathName xview scroll number what

pathName yview ?args?
    pathName yview
    pathName yview moveto fraction
    pathName yview scroll number what

OVERVIEW OF ITEM TYPES
COMMON ITEM OPTIONS
  -dash pattern
  -activedash pattern
  -disableddash pattern
  -dashoffset offset
  -fill color
  -activefill color
  -disabledfill color
  -outline color
  -activeoutline color
  -disabledoutline color
  -offset offset
  -outlinestipple bitmap
  -activeoutlinestipple bitmap
  -disabledoutlinestipple bitmap
  -outlineoffset offset
  -stipple bitmap
  -activestipple bitmap
  -disabledstipple bitmap
  -state state
  -tags tagList
  -width outlineWidth
  -activewidth outlineWidth
  -disabledwidth outlineWidth

ARC ITEMS
  -extent degrees
  -start degrees
  -style type
BITMAP ITEMS
- anchor anchorPos
- background color
- activebackground bitmap
- disabledbackground bitmap
- bitmap bitmap
- activebitmap bitmap
- disabledbitmap bitmap
- foreground color
- activeforeground bitmap
- disabledforeground bitmap

IMAGE ITEMS
- anchor anchorPos
- image name
- activeimage name
- disabledimage name

LINE ITEMS
- arrow where
- arrowshape shape
- capstyle style
- joinstyle style
- smooth smoothMethod
- splinesteps number

OVAL ITEMS

POLYGON ITEMS
- joinstyle style
- smooth boolean
- splinesteps number

RECTANGLE ITEMS

TEXT ITEMS
- anchor anchorPos
- font fontName
- justify how
- text string
- underline
- width lineLength

WINDOW ITEMS
-anchor anchorPos
-height pixels
-width pixels
-window pathName

APPLICATION-DEFINED ITEM TYPES

BINDINGS
CREDITS
SEE ALSO
KEYWORDS

---

**NAME**

canvas - Create and manipulate canvas widgets

---

**SYNOPSIS**

canvas *pathName* ?*options*?

---

**STANDARD OPTIONS**

- background or -bg, background, Background
- borderwidth or -bd, borderWidth, BorderWidth
- cursor, cursor, Cursor
- highlightbackground, highlightBackground, HighlightBackground
- highlightcolor, highlightColor, HighlightColor
- highlightthickness, highlightThickness, HighlightThickness
- insertbackground, insertBackground, Foreground
- insertborderwidth, insertBorderWidth, BorderWidth
- insertofftime, insertOffTime, OffTime
- insertontime, insertOnTime, OnTime
- insertwidth, insertWidth, InsertWidth
- relief, relief, Relief
- selectbackground, selectBackground, Foreground
- selectborderwidth, selectBorderWidth, BorderWidth
- selectforeground, selectForeground, Background
- takefocus, takeFocus, TakeFocus
- xscrollcommand, xScrollCommand, ScrollCommand
- yscrollcommand, yScrollCommand, ScrollCommand
**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: `-closeenough`
Database Name: `closeEnough`
Database Class: `CloseEnough`
  Specifies a floating-point value indicating how close the mouse cursor must be to an item before it is considered to be “inside” the item. Defaults to 1.0.

Command-Line Name: `-confine`
Database Name: `confine`
Database Class: `Confine`
  Specifies a boolean value that indicates whether or not it should be allowable to set the canvas's view outside the region defined by the `scrollRegion` argument. Defaults to true, which means that the view will be constrained within the scroll region.

Command-Line Name: `-height`
Database Name: `height`
Database Class: `Height`
  Specifies a desired window height that the canvas widget should request from its geometry manager. The value may be specified in any of the forms described in the **COORDINATES** section below.

Command-Line Name: `-scrollregion`
Database Name: `scrollRegion`
Database Class: `ScrollRegion`
  Specifies a list with four coordinates describing the left, top, right, and bottom coordinates of a rectangular region. This region is used for scrolling purposes and is considered to be the boundary of the information in the canvas. Each of the coordinates may be specified in any of the forms given in the **COORDINATES** section below.

Command-Line Name: `-state`
Database Name: `state`
Database Class: `State`
  Modifies the default state of the canvas where `state` may be set to
one of: normal, disabled, or hidden. Individual canvas objects all have their own state option which may override the default state. Many options can take separate specifications such that the appearance of the item can be different in different situations. The options that start with active control the appearance when the mouse pointer is over it, while the option starting with disabled controls the appearance when the state is disabled. Canvas items which are disabled will not react to canvas bindings.

Command-Line Name: -width
Database Name: width
Database Class: width
   Specifies a desired window width that the canvas widget should request from its geometry manager. The value may be specified in any of the forms described in the COORDINATES section below.

Command-Line Name: -xscrollincrement
Database Name: xScrollIncrement
Database Class: ScrollIncrement
   Specifies an increment for horizontal scrolling, in any of the usual forms permitted for screen distances. If the value of this option is greater than zero, the horizontal view in the window will be constrained so that the canvas x coordinate at the left edge of the window is always an even multiple of xScrollIncrement; furthermore, the units for scrolling (e.g., the change in view when the left and right arrows of a scrollbar are selected) will also be xScrollIncrement. If the value of this option is less than or equal to zero, then horizontal scrolling is unconstrained.

Command-Line Name: -yscrollincrement
Database Name: yScrollIncrement
Database Class: ScrollIncrement
   Specifies an increment for vertical scrolling, in any of the usual forms permitted for screen distances. If the value of this option is greater than zero, the vertical view in the window will be constrained so that the canvas y coordinate at the top edge of the window is always an even multiple of yScrollIncrement; furthermore, the units for scrolling (e.g., the change in view when
The top and bottom arrows of a scrollbar are selected) will also be yScrollIncrement. If the value of this option is less than or equal to zero, then vertical scrolling is unconstrained.

INTRODUCTION

The canvas command creates a new window (given by the pathName argument) and makes it into a canvas widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the canvas such as its colors and 3-D relief. The canvas command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

Canvas widgets implement structured graphics. A canvas displays any number of items, which may be things like rectangles, circles, lines, and text. Items may be manipulated (e.g. moved or re-colored) and commands may be associated with items in much the same way that the bind command allows commands to be bound to widgets. For example, a particular command may be associated with the <Button-1> event so that the command is invoked whenever button 1 is pressed with the mouse cursor over an item. This means that items in a canvas can have behaviors defined by the Tcl scripts bound to them.

DISPLAY LIST

The items in a canvas are ordered for purposes of display, with the first item in the display list being displayed first, followed by the next item in the list, and so on. Items later in the display list obscure those that are earlier in the display list and are sometimes referred to as being “on top” of earlier items. When a new item is created it is placed at the end of the display list, on top of everything else. Widget commands may be used to re-arrange the order of the display list.

Window items are an exception to the above rules. The underlying window systems require them always to be drawn on top of other items. In addition, the stacking order of window items is not affected by any of the canvas widget commands; you must use the raise and lower Tk
commands instead.

## ITEM IDS AND TAGS

Items in a canvas widget may be named in either of two ways: by id or by tag. Each item has a unique identifying number, which is assigned to that item when it is created. The id of an item never changes and id numbers are never re-used within the lifetime of a canvas widget.

Each item may also have any number of *tags* associated with it. A tag is just a string of characters, and it may take any form except that of an integer. For example, “x123” is OK but “123” is not. The same tag may be associated with many different items. This is commonly done to group items in various interesting ways; for example, all selected items might be given the tag “selected”.

The tag **all** is implicitly associated with every item in the canvas; it may be used to invoke operations on all the items in the canvas.

The tag **current** is managed automatically by Tk; it applies to the *current item*, which is the topmost item whose drawn area covers the position of the mouse cursor (different item types interpret this in varying ways; see the individual item type documentation for details). If the mouse is not in the canvas widget or is not over an item, then no item has the **current** tag.

When specifying items in canvas widget commands, if the specifier is an integer then it is assumed to refer to the single item with that id. If the specifier is not an integer, then it is assumed to refer to all of the items in the canvas that have a tag matching the specifier. The symbol **tagOrId** is used below to indicate that an argument specifies either an id that selects a single item or a tag that selects zero or more items.

**tagOrId** may contain a logical expressions of tags by using operators: “&&”, “||”, “^”, “!”, and parenthesized subexpressions. For example:

```
c find withtag {(a&&!b)||(!a&&b)}
```
or equivalently:

```
c find withtag {a^b}
```

will find only those items with either “a” or “b” tags, but not both.

Some widget commands only operate on a single item at a time; if `tagOrId` is specified in a way that names multiple items, then the normal behavior is for the command to use the first (lowest) of these items in the display list that is suitable for the command. Exceptions are noted in the widget command descriptions below.

**COORDINATES**

All coordinates related to canvases are stored as floating-point numbers. Coordinates and distances are specified in screen units, which are floating-point numbers optionally followed by one of several letters. If no letter is supplied then the distance is in pixels. If the letter is `m` then the distance is in millimeters on the screen; if it is `c` then the distance is in centimeters; `i` means inches, and `p` means printers points (1/72 inch). Larger y-coordinates refer to points lower on the screen; larger x-coordinates refer to points farther to the right. Coordinates can be specified either as an even number of parameters, or as a single list parameter containing an even number of x and y coordinate values.

**TRANSFORMATIONS**

Normally the origin of the canvas coordinate system is at the upper-left corner of the window containing the canvas. It is possible to adjust the origin of the canvas coordinate system relative to the origin of the window using the `xview` and `yview` widget commands; this is typically used for scrolling. Canvases do not support scaling or rotation of the canvas coordinate system relative to the window coordinate system.

Individual items may be moved or scaled using widget commands
described below, but they may not be rotated.

Note that the default origin of the canvas's visible area is coincident with the origin for the whole window as that makes bindings using the mouse position easier to work with; you only need to use the `canvasx` and `canvasy` widget commands if you adjust the origin of the visible area. However, this also means that any focus ring (as controlled by the `-highlightthickness` option) and window border (as controlled by the `-borderwidth` option) must be taken into account before you get to the visible area of the canvas.

**INDICES**

Text items support the notion of an *index* for identifying particular positions within the item. In a similar fashion, line and polygon items support *index* for identifying, inserting and deleting subsets of their coordinates. Indices are used for commands such as inserting or deleting a range of characters or coordinates, and setting the insertion cursor position. An index may be specified in any of a number of ways, and different types of items may support different forms for specifying indices. Text items support the following forms for an index; if you define new types of text-like items, it would be advisable to support as many of these forms as practical. Note that it is possible to refer to the character just after the last one in the text item; this is necessary for such tasks as inserting new text at the end of the item. Lines and Polygons do not support the insertion cursor and the selection. Their indices are supposed to be even always, because coordinates always appear in pairs.

*number*

A decimal number giving the position of the desired character within the text item. 0 refers to the first character, 1 to the next character, and so on. If indexes are odd for lines and polygons, they will be automatically decremented by one. A number less than 0 is treated as if it were zero, and a number greater than the length of the text item is treated as if it were equal to the length of the text item. For polygons, numbers less than 0 or greater then the length of the coordinate list will be adjusted by adding or subtracting the
length until the result is between zero and the length, inclusive.

end
  Refers to the character or coordinate just after the last one in the item (same as the number of characters or coordinates in the item).

insert
  Refers to the character just before which the insertion cursor is drawn in this item. Not valid for lines and polygons.

sel.first
  Refers to the first selected character in the item. If the selection is not in this item then this form is illegal.

sel.last
  Refers to the last selected character in the item. If the selection is not in this item then this form is illegal.

@x,y
  Refers to the character or coordinate at the point given by x and y, where x and y are specified in the coordinate system of the canvas. If x and y lie outside the coordinates covered by the text item, then they refer to the first or last character in the line that is closest to the given point.

DASH PATTERNS

Many items support the notion of a dash pattern for outlines.

The first possible syntax is a list of integers. Each element represents the number of pixels of a line segment. Only the odd segments are drawn using the “outline” color. The other segments are drawn transparent.

The second possible syntax is a character list containing only 5 possible characters “.,-”. The space can be used to enlarge the space between other line elements, and cannot occur as the first position in the string. Some examples:
The main difference of this syntax with the previous is that it is shape-conserving. This means that all values in the dash list will be multiplied by the line width before display. This assures that “.” will always be displayed as a dot and “-” always as a dash regardless of the line width.

On systems which support only a limited set of dash patterns, the dash pattern will be displayed as the closest dash pattern that is available. For example, on Windows only the first 4 of the above examples are available. The last 2 examples will be displayed identically to the first one.

### WIDGET COMMAND

The **canvas** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following widget commands are possible for canvas widgets:

```
pathName addtag tag searchSpec ?arg arg ...?
```

For each item that meets the constraints specified by *searchSpec* and the *args*, add *tag* to the list of tags associated with the item if it
is not already present on that list. It is possible that no items will satisfy the constraints given by searchSpec and args, in which case the command has no effect. This command returns an empty string as result. SearchSpec and arg's may take any of the following forms:

**above** tagOrId

Selects the item just after (above) the one given by tagOrId in the display list. If tagOrId denotes more than one item, then the last (topmost) of these items in the display list is used.

**all**

Selects all the items in the canvas.

**below** tagOrId

Selects the item just before (below) the one given by tagOrId in the display list. If tagOrId denotes more than one item, then the first (lowest) of these items in the display list is used.

**closest** x y ?halo? ?start?

Selects the item closest to the point given by x and y. If more than one item is at the same closest distance (e.g. two items overlap the point), then the top-most of these items (the last one in the display list) is used. If halo is specified, then it must be a non-negative value. Any item closer than halo to the point is considered to overlap it. The start argument may be used to step circularly through all the closest items. If start is specified, it names an item using a tag or id (if by tag, it selects the first item in the display list with the given tag). Instead of selecting the topmost closest item, this form will select the topmost closest item that is below start in the display list; if no such item exists, then the selection behaves as if the start argument had not been specified.

**enclosed** x1 y1 x2 y2

Selects all the items completely enclosed within the rectangular region given by x1, y1, x2, and y2. x1 must be no greater then x2 and y1 must be no greater than y2.
overlapping \( x_1 \ y_1 \ x_2 \ y_2 \)
Selects all the items that overlap or are enclosed within the rectangular region given by \( x_1, y_1, x_2, \) and \( y_2 \). \( x_1 \) must be no greater then \( x_2 \) and \( y_1 \) must be no greater than \( y_2 \).

withtag \( \text{tagOrId} \)
Selects all the items given by \( \text{tagOrId} \).

\textbf{pathName bbox} \( \text{tagOrId} \ \? \text{tagOrId} \ \text{tagOrId} \ \ldots \ ? \)
Returns a list with four elements giving an approximate bounding box for all the items named by the \( \text{tagOrId} \) arguments. The list has the form \( "x_1 \ y_1 \ x_2 \ y_2" \) such that the drawn areas of all the named elements are within the region bounded by \( x_1 \) on the left, \( x_2 \) on the right, \( y_1 \) on the top, and \( y_2 \) on the bottom. The return value may overestimate the actual bounding box by a few pixels. If no items match any of the \( \text{tagOrId} \) arguments or if the matching items have empty bounding boxes (i.e. they have nothing to display) then an empty string is returned.

\textbf{pathName bind} \( \text{tagOrId} \ \? \text{sequence}\ ? \text{command}\ ? \)
This command associates \( \text{command} \) with all the items given by \( \text{tagOrId} \) such that whenever the event sequence given by \( \text{sequence} \) occurs for one of the items the command will be invoked. This widget command is similar to the \( \text{bind} \) command except that it operates on items in a canvas rather than entire widgets. See the \( \text{bind} \) manual entry for complete details on the syntax of \( \text{sequence} \) and the substitutions performed on \( \text{command} \) before invoking it. If all arguments are specified then a new binding is created, replacing any existing binding for the same \( \text{sequence} \) and \( \text{tagOrId} \) (if the first character of \( \text{command} \) is “+” then \( \text{command} \) augments an existing binding rather than replacing it). In this case the return value is an empty string. If \( \text{command} \) is omitted then the command returns the \( \text{command} \) associated with \( \text{tagOrId} \) and \( \text{sequence} \) (an error occurs if there is no such binding). If both \( \text{command} \) and \( \text{sequence} \) are omitted then the command returns a list of all the sequences for which bindings have been defined for \( \text{tagOrId} \).

The only events for which bindings may be specified are those
related to the mouse and keyboard (such as Enter, Leave, ButtonPress, Motion, and KeyPress) or virtual events. The handling of events in canvases uses the current item defined in ITEM IDS AND TAGS above. Enter and Leave events trigger for an item when it becomes the current item or ceases to be the current item; note that these events are different than Enter and Leave events for windows. Mouse-related events are directed to the current item, if any. Keyboard-related events are directed to the focus item, if any (see the focus widget command below for more on this). If a virtual event is used in a binding, that binding can trigger only if the virtual event is defined by an underlying mouse-related or keyboard-related event.

It is possible for multiple bindings to match a particular event. This could occur, for example, if one binding is associated with the item's id and another is associated with one of the item's tags. When this occurs, all of the matching bindings are invoked. A binding associated with the all tag is invoked first, followed by one binding for each of the item's tags (in order), followed by a binding associated with the item's id. If there are multiple matching bindings for a single tag, then only the most specific binding is invoked. A continue command in a binding script terminates that script, and a break command terminates that script and skips any remaining scripts for the event, just as for the bind command.

If bindings have been created for a canvas window using the bind command, then they are invoked in addition to bindings created for the canvas's items using the bind widget command. The bindings for items will be invoked before any of the bindings for the window as a whole.

\[\text{pathName canvasx screenx ?gridspacing?}\]
Given a window x-coordinate in the canvas screenx, this command returns the canvas x-coordinate that is displayed at that location. If gridspacing is specified, then the canvas coordinate is rounded to the nearest multiple of gridspacing units.

\[\text{pathName canvasy screeny ?gridspacing?}\]
Given a window y-coordinate in the canvas `screeny` this command returns the canvas y-coordinate that is displayed at that location. If `gridspacing` is specified, then the canvas coordinate is rounded to the nearest multiple of `gridspacing` units.

`pathName cget option`
Returns the current value of the configuration option given by `option`. `Option` may have any of the values accepted by the `canvas` command.

`pathName configure ?option? ?value? ?option value ...?`
Query or modify the configuration options of the widget. If no `option` is specified, returns a list describing all of the available options for `pathName` (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option-value` pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. `Option` may have any of the values accepted by the `canvas` command.

`pathName coords tagOrId ?x0 y0 ...?`

`pathName coords tagOrId ?coordList?`
Query or modify the coordinates that define an item. If no coordinates are specified, this command returns a list whose elements are the coordinates of the item named by `tagOrId`. If coordinates are specified, then they replace the current coordinates for the named item. If `tagOrId` refers to multiple items, then the first one in the display list is used.

`pathName create type x y ?x y ...? ?option value ...?`

`pathName create type coordList ?option value ...?`
Create a new item in `pathName` of type `type`. The exact format of the arguments after `type` depends on `type`, but usually they consist
of the coordinates for one or more points, followed by specifications for zero or more item options. See the subsections on individual item types below for more on the syntax of this command. This command returns the id for the new item.

`pathName dchars tagOrId first ?last?`
For each item given by `tagOrId`, delete the characters, or coordinates, in the range given by `first` and `last`, inclusive. If some of the items given by `tagOrId` do not support indexing operations then they ignore dchars. Text items interpret `first` and `last` as indices to a character, line and polygon items interpret them indices to a coordinate (an x,y pair). Indices are described in `INDICES` above. If `last` is omitted, it defaults to `first`. This command returns an empty string.

`pathName delete ?tagOrId tagOrId ...?`
Delete each of the items given by each `tagOrId`, and return an empty string.

`pathName dtag tagOrId ?tagToDelete?`
For each of the items given by `tagOrId`, delete the tag given by `tagToDelete` from the list of those associated with the item. If an item does not have the tag `tagToDelete` then the item is unaffected by the command. If `tagToDelete` is omitted then it defaults to `tagOrId`. This command returns an empty string.

`pathName find searchCommand ?arg arg ...?`
This command returns a list consisting of all the items that meet the constraints specified by `searchCommand` and `arg's`. `SearchCommand` and `args` have any of the forms accepted by the `addtag` command. The items are returned in stacking order, with the lowest item first.

`pathName focus ?tagOrId?`
Set the keyboard focus for the canvas widget to the item given by `tagOrId`. If `tagOrId` refers to several items, then the focus is set to the first such item in the display list that supports the insertion cursor. If `tagOrId` does not refer to any items, or if none of them
support the insertion cursor, then the focus is not changed. If
*tagOrId* is an empty string, then the focus item is reset so that no
item has the focus. If *tagOrId* is not specified then the command
returns the id for the item that currently has the focus, or an empty
string if no item has the focus.

Once the focus has been set to an item, the item will display the
insertion cursor and all keyboard events will be directed to that
item. The focus item within a canvas and the focus window on the
screen (set with the *focus* command) are totally independent: a
given item does not actually have the input focus unless (a) its
canvas is the focus window and (b) the item is the focus item within
the canvas. In most cases it is advisable to follow the *focus* widget
command with the *focus* command to set the focus window to the
canvas (if it was not there already).

\[\textit{pathName gettags tagOrId} \]
Return a list whose elements are the tags associated with the item
given by *tagOrId*. If *tagOrId* refers to more than one item, then the
tags are returned from the first such item in the display list. If
*tagOrId* does not refer to any items, or if the item contains no tags,
then an empty string is returned.

\[\textit{pathName icursor tagOrId index} \]
Set the position of the insertion cursor for the item(s) given by
*tagOrId* to just before the character whose position is given by
*index*. If some or all of the items given by *tagOrId* do not support an
insertion cursor then this command has no effect on them. See
\[\textit{INDICES} \] above for a description of the legal forms for *index*. Note:
the insertion cursor is only displayed in an item if that item currently
has the keyboard focus (see the widget command *focus*, below),
but the cursor position may be set even when the item does not
have the focus. This command returns an empty string.

\[\textit{pathName index tagOrId index} \]
This command returns a decimal string giving the numerical index
within *tagOrId* corresponding to *index*. *Index* gives a textual
description of the desired position as described in \[\textit{INDICES} \] above.
Text items interpret index as an index to a character, line and polygon items interpret it as an index to a coordinate (an x,y pair). The return value is guaranteed to lie between 0 and the number of characters, or coordinates, within the item, inclusive. If tagOrId refers to multiple items, then the index is processed in the first of these items that supports indexing operations (in display list order).

```
pathName insert tagOrId beforeThis string
```
For each of the items given by tagOrId, if the item supports text or coordinate, insertion then string is inserted into the item's text just before the character, or coordinate, whose index is beforeThis. Text items interpret beforeThis as an index to a character, line and polygon items interpret it as an index to a coordinate (an x,y pair). For lines and polygons the string must be a valid coordinate sequence. See INDICES above for information about the forms allowed for beforeThis. This command returns an empty string.

```
pathName itemcget tagOrId option
```
Returns the current value of the configuration option for the item given by tagOrId whose name is option. This command is similar to the cget widget command except that it applies to a particular item rather than the widget as a whole. Option may have any of the values accepted by the create widget command when the item was created. If tagOrId is a tag that refers to more than one item, the first (lowest) such item is used.

```
pathName itemconfigure tagOrId ?option? ?value? ?option value ...?
```
This command is similar to the configure widget command except that it modifies item-specific options for the items given by tagOrId instead of modifying options for the overall canvas widget. If no option is specified, returns a list describing all of the available options for the first item given by tagOrId (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option-value pairs are specified, then the command modifies the given widget option(s) to have the given value(s) in each of the
items given by `tagOrId`; in this case the command returns an empty string. The `options` and `values` are the same as those permissible in the create widget command when the item(s) were created; see the sections describing individual item types below for details on the legal options.

`pathName lower tagOrId ?belowThis?`
Move all of the items given by `tagOrId` to a new position in the display list just before the item given by `belowThis`. If `tagOrId` refers to more than one item then all are moved but the relative order of the moved items will not be changed. `BelowThis` is a tag or id; if it refers to more than one item then the first (lowest) of these items in the display list is used as the destination location for the moved items. Note: this command has no effect on window items. Window items always obscure other item types, and the stacking order of window items is determined by the `raise` and `lower` commands, not the `raise` and `lower` widget commands for canvases. This command returns an empty string.

`pathName move tagOrId xAmount yAmount`
Move each of the items given by `tagOrId` in the canvas coordinate space by adding `xAmount` to the x-coordinate of each point associated with the item and `yAmount` to the y-coordinate of each point associated with the item. This command returns an empty string.

`pathName postscript ?option value option value ...?`
Generate a Postscript representation for part or all of the canvas. If the `-file` option is specified then the Postscript is written to a file and an empty string is returned; otherwise the Postscript is returned as the result of the command. If the interpreter that owns the canvas is marked as safe, the operation will fail because safe interpreters are not allowed to write files. If the `-channel` option is specified, the argument denotes the name of a channel already opened for writing. The Postscript is written to that channel, and the channel is left open for further writing at the end of the operation. The Postscript is created in Encapsulated Postscript form using version 3.0 of the Document Structuring Conventions. Note: by
default Postscript is only generated for information that appears in the canvas's window on the screen. If the canvas is freshly created it may still have its initial size of 1x1 pixel so nothing will appear in the Postscript. To get around this problem either invoke the update command to wait for the canvas window to reach its final size, or else use the -width and -height options to specify the area of the canvas to print. The option-value argument pairs provide additional information to control the generation of Postscript. The following options are supported:

- **colormap varName**
  VarName must be the name of an array variable that specifies a color mapping to use in the Postscript. Each element of varName must consist of Postscript code to set a particular color value (e.g. “1.0 1.0 0.0 setrgbcolor”). When outputting color information in the Postscript, Tk checks to see if there is an element of varName with the same name as the color. If so, Tk uses the value of the element as the Postscript command to set the color. If this option has not been specified, or if there is no entry in varName for a given color, then Tk uses the red, green, and blue intensities from the X color.

- **colormode mode**
  Specifies how to output color information. Mode must be either color (for full color output), gray (convert all colors to their gray-scale equivalents) or mono (convert all colors to black or white).

- **file fileName**
  Specifies the name of the file in which to write the Postscript. If this option is not specified then the Postscript is returned as the result of the command instead of being written to a file.

- **fontmap varName**
  VarName must be the name of an array variable that specifies a font mapping to use in the Postscript. Each element of varName must consist of a Tcl list with two elements, which are the name and point size of a Postscript font. When outputting
Postscript commands for a particular font, Tk checks to see if `varName` contains an element with the same name as the font. If there is such an element, then the font information contained in that element is used in the Postscript. Otherwise Tk attempts to guess what Postscript font to use. Tk's guesses generally only work for well-known fonts such as Times and Helvetica and Courier, and only if the X font name does not omit any dashes up through the point size. For example, `- *- Courier-Bold-R-Normal--*-120-*` will work but `*Courier-Bold-R-Normal*120*` will not; Tk needs the dashes to parse the font name).

- **height size**
  Specifies the height of the area of the canvas to print. Defaults to the height of the canvas window.

- **pageanchor anchor**
  Specifies which point of the printed area of the canvas should appear over the positioning point on the page (which is given by the `-pagex` and `-pagey` options). For example, `-pageanchor n` means that the top center of the area of the canvas being printed (as it appears in the canvas window) should be over the positioning point. Defaults to `center`.

- **pageheight size**
  Specifies that the Postscript should be scaled in both x and y so that the printed area is `size` high on the Postscript page. `Size` consists of a floating-point number followed by `c` for centimeters, `i` for inches, `m` for millimeters, or `p` or nothing for printer's points (1/72 inch). Defaults to the height of the printed area on the screen. If both `-pageheight` and `-pagewidth` are specified then the scale factor from `-pagewidth` is used (non-uniform scaling is not implemented).

- **pagewidth size**
  Specifies that the Postscript should be scaled in both x and y so that the printed area is `size` wide on the Postscript page. `Size` has the same form as for `-pageheight`. Defaults to the
width of the printed area on the screen. If both -pageheight and -pagewidth are specified then the scale factor from -pagewidth is used (non-uniform scaling is not implemented).

**-pagex position**

*Position* gives the x-coordinate of the positioning point on the Postscript page, using any of the forms allowed for -pageheight. Used in conjunction with the -pagey and -pageanchor options to determine where the printed area appears on the Postscript page. Defaults to the center of the page.

**-pagey position**

*Position* gives the y-coordinate of the positioning point on the Postscript page, using any of the forms allowed for -pageheight. Used in conjunction with the -pagex and -pageanchor options to determine where the printed area appears on the Postscript page. Defaults to the center of the page.

**-rotate boolean**

*Boolean* specifies whether the printed area is to be rotated 90 degrees. In non-rotated output the x-axis of the printed area runs along the short dimension of the page ("portrait"orientation); in rotated output the x-axis runs along the long dimension of the page ("landscape"orientation). Defaults to non-rotated.

**-width size**

Specifies the width of the area of the canvas to print. Defaults to the width of the canvas window.

**-x position**

Specifies the x-coordinate of the left edge of the area of the canvas that is to be printed, in canvas coordinates, not window coordinates. Defaults to the coordinate of the left edge of the window.
-y position
   Specifies the y-coordinate of the top edge of the area of the canvas that is to be printed, in canvas coordinates, not window coordinates. Defaults to the coordinate of the top edge of the window.

pathName raise tagOrId ?aboveThis?
   Move all of the items given by tagOrId to a new position in the display list just after the item given by aboveThis. If tagOrId refers to more than one item then all are moved but the relative order of the moved items will not be changed. AboveThis is a tag or id; if it refers to more than one item then the last (topmost) of these items in the display list is used as the destination location for the moved items. Note: this command has no effect on window items. Window items always obscure other item types, and the stacking order of window items is determined by the raise and lower commands, not the raise and lower widget commands for canvases. This command returns an empty string.

pathName scale tagOrId xOrigin yOrigin xScale yScale
   Rescale all of the items given by tagOrId in canvas coordinate space. XOrigin and yOrigin identify the origin for the scaling operation and xScale and yScale identify the scale factors for x- and y-coordinates, respectively (a scale factor of 1.0 implies no change to that coordinate). For each of the points defining each item, the x-coordinate is adjusted to change the distance from xOrigin by a factor of xScale. Similarly, each y-coordinate is adjusted to change the distance from yOrigin by a factor of yScale. This command returns an empty string.

pathName scan option args
   This command is used to implement scanning on canvases. It has two forms, depending on option:

pathName scan mark x y
   Records x and y and the canvas's current view; used in conjunction with later scan dragto commands. Typically this command is associated with a mouse button press in the
widget and x and y are the coordinates of the mouse. It returns an empty string.

\texttt{pathname scan dragto x y \?gain\?.}
This command computes the difference between its x and y arguments (which are typically mouse coordinates) and the x and y arguments to the last \texttt{scan mark} command for the widget. It then adjusts the view by \textit{gain} times the difference in coordinates, where \textit{gain} defaults to 10. This command is typically associated with mouse motion events in the widget, to produce the effect of dragging the canvas at high speed through its window. The return value is an empty string.

\texttt{pathname select option \?tagOrId arg?}
Manipulates the selection in one of several ways, depending on \textit{option}. The command may take any of the forms described below. In all of the descriptions below, \textit{tagOrId} must refer to an item that supports indexing and selection; if it refers to multiple items then the first of these that supports indexing and the selection is used. \textit{Index} gives a textual description of a position within \textit{tagOrId}, as described in \texttt{INDICES} above.

\texttt{pathname select adjust tagOrId index}
Locate the end of the selection in \textit{tagOrId} nearest to the character given by \textit{index}, and adjust that end of the selection to be at \textit{index} (i.e. including but not going beyond \textit{index}). The other end of the selection is made the anchor point for future \texttt{select to} commands. If the selection is not currently in \textit{tagOrId} then this command behaves the same as the \texttt{select to} widget command. Returns an empty string.

\texttt{pathname select clear}
Clear the selection if it is in this widget. If the selection is not in this widget then the command has no effect. Returns an empty string.

\texttt{pathname select from tagOrId index}
Set the selection anchor point for the widget to be just before
the character given by \textit{index} in the item given by \textit{tagOrId}. This command does not change the selection; it just sets the fixed end of the selection for future \texttt{select to} commands. Returns an empty string.

\textit{pathName select item}
Returns the id of the selected item, if the selection is in an item in this canvas. If the selection is not in this canvas then an empty string is returned.

\textit{pathName select to tagOrId index}
Set the selection to consist of those characters of \textit{tagOrId} between the selection anchor point and \textit{index}. The new selection will include the character given by \textit{index}; it will include the character given by the anchor point only if \textit{index} is greater than or equal to the anchor point. The anchor point is determined by the most recent \texttt{select adjust} or \texttt{select from} command for this widget. If the selection anchor point for the widget is not currently in \textit{tagOrId}, then it is set to the same character given by \textit{index}. Returns an empty string.

\textit{pathName type tagOrId}
Returns the type of the item given by \textit{tagOrId}, such as \texttt{rectangle} or \texttt{text}. If \textit{tagOrId} refers to more than one item, then the type of the first item in the display list is returned. If \textit{tagOrId} does not refer to any items at all then an empty string is returned.

\textit{pathName xview ?args?}
This command is used to query and change the horizontal position of the information displayed in the canvas's window. It can take any of the following forms:

\textit{pathName xview}
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20\% of the canvas's area (as defined by the \texttt{-scrollregion} option) is off-
screen to the left, the middle 40% is visible in the window, and 40% of the canvas is off-screen to the right. These are the same values passed to scrollbars via the -xscrollcommand option.

```
pathName xview moveto fraction
```
Adjusts the view in the window so that fraction of the total width of the canvas is off-screen to the left. Fraction must be a fraction between 0 and 1.

```
pathName xview scroll number what
```
This command shifts the view in the window left or right according to number and what. Number must be an integer. What must be either units or pages or an abbreviation of one of these. If what is units, the view adjusts left or right in units of the xScrollIncrement option, if it is greater than zero, or in units of one-tenth the window's width otherwise. If what is pages then the view adjusts in units of nine-tenths the window's width. If number is negative then information farther to the left becomes visible; if it is positive then information farther to the right becomes visible.

```
pathName yview ?args?
```
This command is used to query and change the vertical position of the information displayed in the canvas's window. It can take any of the following forms:

```
pathName yview
```
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the vertical span that is visible in the window. For example, if the first element is .6 and the second element is 1.0, the lowest 40% of the canvas's area (as defined by the -scrollregion option) is visible in the window. These are the same values passed to scrollbars via the -yscrollcommand option.

```
pathName yview moveto fraction
```
Adjusts the view in the window so that fraction of the canvas's
area is off-screen to the top. *Fraction* is a fraction between 0 and 1.

*pathName yview scroll number what*
This command adjusts the view in the window up or down according to *number* and *what*. *Number* must be an integer. *What* must be either *units* or *pages*. If *what* is *units*, the view adjusts up or down in units of the *yScrollIncrement* option, if it is greater than zero, or in units of one-tenth the window's height otherwise. If *what* is *pages* then the view adjusts in units of nine-tenths the window's height. If *number* is negative then higher information becomes visible; if it is positive then lower information becomes visible.

### OVERVIEW OF ITEM TYPES

The sections below describe the various types of items supported by canvas widgets. Each item type is characterized by two things: first, the form of the *create* command used to create instances of the type; and second, a set of configuration options for items of that type, which may be used in the *create* and *itemconfigure* widget commands. Most items do not support indexing or selection or the commands related to them, such as *index* and *insert*. Where items do support these facilities, it is noted explicitly in the descriptions below. At present, text, line and polygon items provide this support. For lines and polygons the indexing facility is used to manipulate the coordinates of the item.

### COMMON ITEM OPTIONS

Many items share a common set of options. These options are explained here, and then referred to be each widget type for brevity.

- **-dash pattern**
- **-activedash pattern**
- **-disableddash pattern**
  This option specifies dash patterns for the normal, active state, and
disabled state of an item. pattern may have any of the forms accepted by Tk_GetDash. If the dash options are omitted then the default is a solid outline. See DASH PATTERNS for more information.

-dashoffset offset
The starting offset in pixels into the pattern provided by the -dash option. -dashoffset is ignored if there is no -dash pattern. The offset may have any of the forms described in the COORDINATES section above.

-fill color

-activefill color

-disabledfill color
Specifies the color to be used to fill item's area. in its normal, active, and disabled states, Color may have any of the forms accepted by Tk_GetColor. If color is an empty string (the default), then the item will not be filled. For the line item, it specifies the color of the line drawn. For the text item, it specifies the foreground color of the text.

-outline color

-activeoutline color

-disabledoutline color
This option specifies the color that should be used to draw the outline of the item in its normal, active and disabled states. Color may have any of the forms accepted by Tk_GetColor. This option defaults to black. If color is specified as an empty string then no outline is drawn for the item.

-offset offset
Specifies the offset of stipples. The offset value can be of the form x,y or side, where side can be n, ne, e, se, s, sw, w, nw, or center. In the first case the origin is the origin of the toplevel of the
current window. For the canvas itself and canvas objects the origin is the canvas origin, but putting # in front of the coordinate pair indicates using the toplevel origin instead. For canvas objects, the -offset option is used for stippling as well. For the line and polygon canvas items you can also specify an index as argument, which connects the stipple origin to one of the coordinate points of the line/polygon.

-outlinestipple bitmap

-activeoutlinestipple bitmap

-disabledoutlinestipple bitmap

This option specifies stipple patterns that should be used to draw the outline of the item in its normal, active and disabled states. Indicates that the outline for the item should be drawn with a stipple pattern; bitmap specifies the stipple pattern to use, in any of the forms accepted by Tk_GetBitmap. If the -outline option has not been specified then this option has no effect. If bitmap is an empty string (the default), then the outline is drawn in a solid fashion. Note that stipples are not well supported on platforms that do not use X11 as their drawing API.

-outlineoffset offset

Specifies the offset of the stipple pattern used for outlines. The offset value can be of the form “x,y” or the description of a side (one of n, ne, e, se, s, sw, w, nw, or center). This option only has an effect when the outline is drawn as a stipple pattern, and is only supported under X11.

-stipple bitmap

-activestipple bitmap

-disabledstipple bitmap

This option specifies stipple patterns that should be used to fill the item in its normal, active and disabled states. bitmap specifies the stipple pattern to use, in any of the forms accepted by
**Tk::GetBitmap.** If the `-fill` option has not been specified then this option has no effect. If `bitmap` is an empty string (the default), then filling is done in a solid fashion. For the text item, it affects the actual text. *Note that stipplers are not well supported on platforms that do not use X11 as their drawing API.*

-**state state**
  This allows an item to override the canvas widget’s global `state` option. It takes the same values: `normal`, `disabled` or `hidden`.

-**tags tagList**
  Specifies a set of tags to apply to the item. `TagList` consists of a list of tag names, which replace any existing tags for the item. `TagList` may be an empty list.

-**width outlineWidth**
-**activewidth outlineWidth**
-**disabledwidth outlineWidth**
  Specifies the width of the outline to be drawn around the item’s region, in its normal, active and disabled states. `outlineWidth` may be in any of the forms described in the `COORDINATES` section above. If the `-outline` option has been specified as an empty string then this option has no effect. This option defaults to 1.0. For arcs, wide outlines will be drawn centered on the edges of the arc’s region.

---

**ARC ITEMS**

Items of type `arc` appear on the display as arc-shaped regions. An arc is a section of an oval delimited by two angles (specified by the `-start` and `-extent` options) and displayed in one of several ways (specified by the `-style` option). Arcs are created with widget commands of the following form:

```
p.pathname create arc x1 y1 x2 y2 ?option value option
```
The arguments \(x_1, y_1, x_2, \) and \(y_2\) or \(coordList\) give the coordinates of two diagonally opposite corners of a rectangular region enclosing the oval that defines the arc. After the coordinates there may be any number of \(option-value\) pairs, each of which sets one of the configuration options for the item. These same \(option-value\) pairs may be used in \texttt{itemconfigure} widget commands to change the item's configuration. An arc item becomes the current item when the mouse pointer is over any part that is painted or (when fully transparent) that would be painted if both the \texttt{-fill} and \texttt{-outline} options were non-empty.

The following standard options are supported by arcs:

- \texttt{-dash}
- \texttt{-activedash}
- \texttt{-disableddash}
- \texttt{-dashoffset}
- \texttt{-fill}
- \texttt{-activefill}
- \texttt{-disabledfill}
- \texttt{-offset}
- \texttt{-outline}
- \texttt{-activeoutline}
- \texttt{-disabledoutline}
- \texttt{-outlineoffset}
- \texttt{-outlinestipple}
- \texttt{-activeoutlinestipple}
- \texttt{-disabledoutlinestipple}
- \texttt{-stipple}
- \texttt{-activestipple}
- \texttt{-disabledstipple}
- \texttt{-state}
- \texttt{-tags}
- \texttt{-width}
The following extra options are supported for arcs:

- **-extent degrees**
  Specifies the size of the angular range occupied by the arc. The arc's range extends for degrees degrees counter-clockwise from the starting angle given by the -start option. Degrees may be negative. If it is greater than 360 or less than -360, then degrees modulo 360 is used as the extent.

- **-start degrees**
  Specifies the beginning of the angular range occupied by the arc. Degrees is given in units of degrees measured counter-clockwise from the 3-o'clock position; it may be either positive or negative.

- **-style type**
  Specifies how to draw the arc. If type is pieslice (the default) then the arc's region is defined by a section of the oval's perimeter plus two line segments, one between the center of the oval and each end of the perimeter section. If type is chord then the arc's region is defined by a section of the oval's perimeter plus a single line segment connecting the two end points of the perimeter section. If type is arc then the arc's region consists of a section of the perimeter alone. In this last case the -fill option is ignored.

### BITMAP ITEMS

Items of type bitmap appear on the display as images with two colors, foreground and background. Bitmaps are created with widget commands of the following form:

```
pathName create bitmap x y ?option value option value ...
pathName create bitmap coordList ?option value option value ...
```
The arguments \(x\) and \(y\) or \textit{coordList}\ (which must have two elements) specify the coordinates of a point used to position the bitmap on the display (see the \texttt{-anchor} option below for more information on how bitmaps are displayed). After the coordinates there may be any number of \textit{option-value} pairs, each of which sets one of the configuration options for the item. These same \textit{option-value} pairs may be used in \texttt{itemconfigure} widget commands to change the item's configuration. A bitmap item becomes the current item when the mouse pointer is over any part of its bounding box.

The following standard options are supported by bitmaps:

\begin{center}
\begin{tabular}{l}
\texttt{-state} \\
\texttt{-tags}
\end{tabular}
\end{center}

The following extra options are supported for bitmaps:

\texttt{-anchor anchorPos}

\textit{AnchorPos} tells how to position the bitmap relative to the positioning point for the item; it may have any of the forms accepted by \texttt{Tk\_GetAnchor}. For example, if \textit{anchorPos} is \texttt{center} then the bitmap is centered on the point; if \textit{anchorPos} is \texttt{n} then the bitmap will be drawn so that its top center point is at the positioning point. This option defaults to \texttt{center}.

\texttt{-background color}

\texttt{-activebackground bitmap}

\texttt{-disabledbackground bitmap}

Specifies the color to use for each of the bitmap's “0” valued pixels in its normal, active and disabled states. \textit{Color} may have any of the forms accepted by \texttt{Tk\_GetColor}. If this option is not specified, or if it is specified as an empty string, then nothing is displayed where the bitmap pixels are 0; this produces a transparent effect.

\texttt{-bitmap bitmap}
-**activebitmap** *bitmap*

-**disabledbitmap** *bitmap*
   Specifies the bitmaps to display in the item in its normal, active and disabled states. *Bitmap* may have any of the forms accepted by [Tk.GetBitmap](#).

-**foreground** *color*

-**activeforeground** *bitmap*

-**disabledforeground** *bitmap*
   Specifies the color to use for each of the bitmap's “1” valued pixels in its normal, active and disabled states. *Color* may have any of the forms accepted by [Tk.GetColor](#) and defaults to black.

### IMAGE ITEMS

Items of type **image** are used to display images on a canvas. Images are created with widget commands of the following form:

```
pathName create image x y ?option value option value ...
```

The arguments *x* and *y* or *coordList* specify the coordinates of a point used to position the image on the display (see the **-anchor** option below for more information). After the coordinates there may be any number of **option-value** pairs, each of which sets one of the configuration options for the item. These same **option-value** pairs may be used in **itemconfigure** widget commands to change the item's configuration. An image item becomes the current item when the mouse pointer is over any part of its bounding box.

The following standard options are supported by images:
The following extra options are supported for images:

**anchor anchorPos**

*AnchorPos* tells how to position the image relative to the positioning point for the item; it may have any of the forms accepted by `Tk_GetAnchor`. For example, if *anchorPos* is *center* then the image is centered on the point; if *anchorPos* is *n* then the image will be drawn so that its top center point is at the positioning point. This option defaults to *center*.

**image name**

**activeimage name**

**disabledimage name**

Specifies the name of the images to display in the item in is normal, active and disabled states. This image must have been created previously with the *image create* command.

**LINE ITEMS**

Items of type *line* appear on the display as one or more connected line segments or curves. Line items support coordinate indexing operations using the canvas widget commands: *dchars, index, insert*. Lines are created with widget commands of the following form:

```
p create line x1 y1... xn yn ?option value option...
p create line coordList ?option value option
```

The arguments *x1* through *yn* or *coordList* give the coordinates for a series of two or more points that describe a series of connected line
segments. After the coordinates there may be any number of option-value pairs, each of which sets one of the configuration options for the item. These same option-value pairs may be used in itemconfigure widget commands to change the item’s configuration. A line item is the current item whenever the mouse pointer is over any segment of the line, whether drawn or not and whether or not the line is smoothed.

The following standard options are supported by lines:

- dash
- activedash
- disableddash
- dashoffset
- fill
- activefill
- disabledfill
- stipple
- activestipple
- disabledstipple
- state
- tags
- width
- activewidth
- disabledwidth

The following extra options are supported for lines:

-arrow where
  Indicates whether or not arrowheads are to be drawn at one or both ends of the line. Where must have one of the values none (for no arrowheads), first (for an arrowhead at the first point of the line), last (for an arrowhead at the last point of the line), or both (for arrowheads at both ends). This option defaults to none.

-arrowshape shape
  This option indicates how to draw arrowheads. The shape
argument must be a list with three elements, each specifying a
distance in any of the forms described in the COORDINATES
section above. The first element of the list gives the distance along
the line from the neck of the arrowhead to its tip. The second
element gives the distance along the line from the trailing points of
the arrowhead to the tip, and the third element gives the distance
from the outside edge of the line to the trailing points. If this option
is not specified then Tk picks a “reasonable” shape.

-capstyle style
    Specifies the ways in which caps are to be drawn at the endpoints
    of the line. Style may have any of the forms accepted by
    Tk_GetCapStyle (butt, projecting, or round). If this option is not
    specified then it defaults to butt. Where arrowheads are drawn the
cap style is ignored.

-joinstyle style
    Specifies the ways in which joints are to be drawn at the vertices of
    the line. Style may have any of the forms accepted by
    Tk_GetCapStyle (bevel, miter, or round). If this option is not
    specified then it defaults to round. If the line only contains two
    points then this option is irrelevant.

-smooth smoothMethod
    smoothMethod must have one of the forms accepted by
    Tcl_GetBoolean or a line smoothing method. Only true and raw
    are supported in the core (with bezier being an alias for true), but
    more can be added at runtime. If a boolean false value or empty
    string is given, no smoothing is applied. A boolean truth value
    assumes true smoothing. If the smoothing method is true, this
    indicates that the line should be drawn as a curve, rendered as a
    set of quadratic splines: one spline is drawn for the first and second
    line segments, one for the second and third, and so on. Straight-
    line segments can be generated within a curve by duplicating the
    end-points of the desired line segment. If the smoothing method is
    raw, this indicates that the line should also be drawn as a curve but
    where the list of coordinates is such that the first coordinate pair
    (and every third coordinate pair thereafter) is a knot point on a
cubic Bezier curve, and the other coordinates are control points on the cubic Bezier curve. Straight line segments can be generated within a curve by making control points equal to their neighbouring knot points. If the last point is a control point and not a knot point, the point is repeated (one or two times) so that it also becomes a knot point.

-splinesteps number
Specifies the degree of smoothness desired for curves: each spline will be approximated with number line segments. This option is ignored unless the -smooth option is true or raw.

## OVAL ITEMS

Items of type oval appear as circular or oval regions on the display. Each oval may have an outline, a fill, or both. Ovals are created with widget commands of the following form:

```
pathName create oval x1 y1 x2 y2 ?option value option value...
```
```
pathName create oval coordList ?option value option value...
```

The arguments \(x1, y1, x2,\) and \(y2\) or coordList give the coordinates of two diagonally opposite corners of a rectangular region enclosing the oval. The oval will include the top and left edges of the rectangle not the lower or right edges. If the region is square then the resulting oval is circular; otherwise it is elongated in shape. After the coordinates there may be any number of option-value pairs, each of which sets one of the configuration options for the item. These same option-value pairs may be used in itemconfigure widget commands to change the item’s configuration. An oval item becomes the current item when the mouse pointer is over any part that is painted or (when fully transparent) that would be painted if both the -fill and -outline options were non-empty.

The following standard options are supported by ovals:
Items of type **polygon** appear as polygonal or curved filled regions on the display. Polygon items support coordinate indexing operations using the canvas widget commands: **dchars**, **index**, **insert**. Polygons are created with widget commands of the following form:

```
pathName create polygon x1 y1 ... xn yn ?option value option value ...
pPathName create polygon coordList ?option value option value ...
```
The arguments $x_1$ through $y_n$ or coordList specify the coordinates for three or more points that define a polygon. The first point should not be repeated as the last to close the shape; Tk will automatically close the periphery between the first and last points. After the coordinates there may be any number of option-value pairs, each of which sets one of the configuration options for the item. These same option-value pairs may be used in itemconfigure widget commands to change the item's configuration. A polygon item is the current item whenever the mouse pointer is over any part of the polygon, whether drawn or not and whether or not the outline is smoothed.

The following standard options are supported by polygons:

<table>
<thead>
<tr>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-dash</td>
</tr>
<tr>
<td>-activedash</td>
</tr>
<tr>
<td>-disableddash</td>
</tr>
<tr>
<td>-dashoffset</td>
</tr>
<tr>
<td>-fill</td>
</tr>
<tr>
<td>-activefill</td>
</tr>
<tr>
<td>-disabledfill</td>
</tr>
<tr>
<td>-offset</td>
</tr>
<tr>
<td>-outline</td>
</tr>
<tr>
<td>-activeoutline</td>
</tr>
<tr>
<td>-disabledoutline</td>
</tr>
<tr>
<td>-outlinestipple</td>
</tr>
<tr>
<td>-activeoutlinestipple</td>
</tr>
<tr>
<td>-disabledoutlinestipple</td>
</tr>
<tr>
<td>-stipple</td>
</tr>
<tr>
<td>-activestipple</td>
</tr>
<tr>
<td>-disabledstipple</td>
</tr>
<tr>
<td>-state</td>
</tr>
<tr>
<td>-tags</td>
</tr>
<tr>
<td>-width</td>
</tr>
<tr>
<td>-activewidth</td>
</tr>
<tr>
<td>-disabledwidth</td>
</tr>
</tbody>
</table>
The following extra options are supported for polygons:

- **joinstyle style**
  Specifies the ways in which joints are to be drawn at the vertices of the outline. *Style* may have any of the forms accepted by `Tk_GetCapStyle` (bevel, miter, or round). If this option is not specified then it defaults to round.

- **smooth boolean**
  *Boolean* must have one of the forms accepted by `Tcl_GetBoolean` or a line smoothing method. Only true and raw are supported in the core (with bezier being an alias for true), but more can be added at runtime. If a boolean false value or empty string is given, no smoothing is applied. A boolean truth value assumes true smoothing. If the smoothing method is true, this indicates that the polygon should be drawn as a curve, rendered as a set of quadratic splines: one spline is drawn for the first and second line segments, one for the second and third, and so on. Straight-line segments can be generated within a curve by duplicating the end-points of the desired line segment. If the smoothing method is raw, this indicates that the polygon should also be drawn as a curve but where the list of coordinates is such that the first coordinate pair (and every third coordinate pair thereafter) is a knot point on a cubic Bezier curve, and the other coordinates are control points on the cubic Bezier curve. Straight line segments can be venerated within a curve by making control points equal to their neighbouring knot points. If the last point is not the second point of a pair of control points, the point is repeated (one or two times) so that it also becomes the second point of a pair of control points (the associated knot point will be the first control point).

- **splinesteps number**
  Specifies the degree of smoothness desired for curves: each spline will be approximated with *number* line segments. This option is ignored unless the -smooth option is true or raw.

Polygon items are different from other items such as rectangles, ovals and arcs in that interior points are considered to be “inside” a polygon
(e.g. for purposes of the **find closest** and **find overlapping** widget commands) even if it is not filled. For most other item types, an interior point is considered to be inside the item only if the item is filled or if it has neither a fill nor an outline. If you would like an unfilled polygon whose interior points are not considered to be inside the polygon, use a line item instead.

## RECTANGLE ITEMS

Items of type **rectangle** appear as rectangular regions on the display. Each rectangle may have an outline, a fill, or both. Rectangles are created with widget commands of the following form:

```
pathName create rectangle x1 y1 x2 y2 ?option value ...
```

The arguments `x1`, `y1`, `x2`, and `y2` or `coordList` (which must have four elements) give the coordinates of two diagonally opposite corners of the rectangle (the rectangle will include its upper and left edges but not its lower or right edges). After the coordinates there may be any number of `option-value` pairs, each of which sets one of the configuration options for the item. These same `option-value` pairs may be used in `itemconfigure` widget commands to change the item's configuration. A rectangle item becomes the current item when the mouse pointer is over any part that is painted or (when fully transparent) that would be painted if both the `-fill` and `-outline` options were non-empty.

The following standard options are supported by rectangles:

- `-dash`
- `-activedash`
- `-disabledash`
- `-dashoffset`
- `-fill`
- `-activefill`
A text item displays a string of characters on the screen in one or more lines. Text items support indexing and selection, along with the following text-related canvas widget commands: **dchars**, **focus**, **icursor**, **index**, **insert**, **select**. Text items are created with widget commands of the following form:

```
pathName create text x y ?option value option value ...
```

The arguments `x` and `y` or `coordList` (which must have two elements) specify the coordinates of a point used to position the text on the display (see the options below for more information on how text is displayed). After the coordinates there may be any number of `option-value` pairs, each of which sets one of the configuration options for the
item. These same *option-value* pairs may be used in `itemconfigure` widget commands to change the item's configuration. A text item becomes the current item when the mouse pointer is over any part of its bounding box.

The following standard options are supported by text items:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fill</td>
<td>Specifies the background color for the text item.</td>
</tr>
<tr>
<td>-activefill</td>
<td>Specifies the background color when the item is active.</td>
</tr>
<tr>
<td>-disabledfill</td>
<td>Specifies the background color when the item is disabled.</td>
</tr>
<tr>
<td>-stipple</td>
<td>Specifies the stipple pattern for the item.</td>
</tr>
<tr>
<td>-activestipple</td>
<td>Specifies the stipple pattern when the item is active.</td>
</tr>
<tr>
<td>-disabledstipple</td>
<td>Specifies the stipple pattern when the item is disabled.</td>
</tr>
<tr>
<td>-state</td>
<td>Specifies the state of the item.</td>
</tr>
<tr>
<td>-tags</td>
<td>Specifies the tags for the item.</td>
</tr>
</tbody>
</table>

The following extra options are supported for text items:

- **anchor anchorPos**
  *AnchorPos* tells how to position the text relative to the positioning point for the text; it may have any of the forms accepted by `Tk_GetAnchor`. For example, if *anchorPos* is `center` then the text is centered on the point; if *anchorPos* is `n` then the text will be drawn such that the top center point of the rectangular region occupied by the text will be at the positioning point. This option defaults to `center`.

- **font fontName**
  Specifies the font to use for the text item. *FontName* may be any string acceptable to `Tk_GetFont`. If this option is not specified, it defaults to a system-dependent font.

- **justify how**
  Specifies how to justify the text within its bounding region. *How* must be one of the values `left`, `right`, or `center`. This option will only matter if the text is displayed as multiple lines. If the option is
omitted, it defaults to left.

-text string
String specifies the characters to be displayed in the text item. Newline characters cause line breaks. The characters in the item may also be changed with the insert and delete widget commands. This option defaults to an empty string.

-underline
Specifies the integer index of a character within the text to be underlined. 0 corresponds to the first character of the text displayed, 1 to the next character, and so on. -1 means that no underline should be drawn (if the whole text item is to be underlined, the appropriate font should be used instead).

-width lineLength
Specifies a maximum line length for the text, in any of the forms described in the COORDINATES section above. If this option is zero (the default) the text is broken into lines only at newline characters. However, if this option is non-zero then any line that would be longer than lineLength is broken just before a space character to make the line shorter than lineLength; the space character is treated as if it were a newline character.

WINDOW ITEMS

Items of type window cause a particular window to be displayed at a given position on the canvas. Window items are created with widget commands of the following form:

```
pathName create window x y ?option value option value ...
pathName create window coordList ?option value option value ...
```

The arguments x and y or coordList (which must have two elements) specify the coordinates of a point used to position the window on the display (see the -anchor option below for more information on how
bitmaps are displayed). After the coordinates there may be any number of *option-value* pairs, each of which sets one of the configuration options for the item. These same *option-value* pairs may be used in *itemconfigure* widget commands to change the item's configuration. Theoretically, a window item becomes the current item when the mouse pointer is over any part of its bounding box, but in practice this typically does not happen because the mouse pointer ceases to be over the canvas at that point.

The following standard options are supported by window items:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-state</td>
<td></td>
</tr>
<tr>
<td>-tags</td>
<td></td>
</tr>
</tbody>
</table>

The following extra options are supported for window items:

- **anchor anchorPos**
  *AnchorPos* tells how to position the window relative to the positioning point for the item; it may have any of the forms accepted by *Tk_GetAnchor*. For example, if *anchorPos* is *center* then the window is centered on the point; if *anchorPos* is *n* then the window will be drawn so that its top center point is at the positioning point. This option defaults to *center*.

- **-height pixels**
  Specifies the height to assign to the item's window. *Pixels* may have any of the forms described in the *COORDINATES* section above. If this option is not specified, or if it is specified as zero, then the window is given whatever height it requests internally.

- **-width pixels**
  Specifies the width to assign to the item's window. *Pixels* may have any of the forms described in the *COORDINATES* section above. If this option is not specified, or if it is specified as zero, then the window is given whatever width it requests internally.
-**window** *pathName*

  Specifies the window to associate with this item. The window specified by *pathName* must either be a child of the canvas widget or a child of some ancestor of the canvas widget. *PathName* may not refer to a top-level window.

Note: due to restrictions in the ways that windows are managed, it is not possible to draw other graphical items (such as lines and images) on top of window items. A window item always obscures any graphics that overlap it, regardless of their order in the display list. Also note that window items, unlike other canvas items, are not clipped for display by their containing canvas's border, and are instead clipped by the parent widget of the window specified by the **-window** option; when the parent widget is the canvas, this means that the window item can overlap the canvas's border.

**APPLICATION-DEFINED ITEM TYPES**

It is possible for individual applications to define new item types for canvas widgets using C code. See the documentation for **Tk_CreateItemType**.

**BINDINGS**

In the current implementation, new canvases are not given any default behavior: you will have to execute explicit Tcl commands to give the canvas its behavior.

**CREDITS**

Tk's canvas widget is a blatant ripoff of ideas from Joel Bartlett's *ezd* program. *Ezd* provides structured graphics in a Scheme environment and preceded canvases by a year or two. Its simple mechanisms for placing and animating graphical objects inspired the functions of canvases.

**SEE ALSO**
NAME

keysyms - keysyms recognized by Tk

DESCRIPTION

Tk recognizes many keysyms when specifying key bindings (e.g. bind . <Key-keysym>). The following list enumerates the keysyms that will be recognized by Tk. Note that not all keysyms will be valid on all platforms. For example, on Unix systems, the presence of a particular keysym is dependant on the configuration of the keyboard modifier map. This list shows keysyms along with their decimal and hexadecimal values.

<table>
<thead>
<tr>
<th>Keysym</th>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>space</td>
<td>32</td>
<td>0x0020</td>
</tr>
<tr>
<td>exclam</td>
<td>33</td>
<td>0x0021</td>
</tr>
<tr>
<td>quotedbl</td>
<td>34</td>
<td>0x0022</td>
</tr>
<tr>
<td>numbersign</td>
<td>35</td>
<td>0x0023</td>
</tr>
<tr>
<td>dollar</td>
<td>36</td>
<td>0x0024</td>
</tr>
<tr>
<td>percent</td>
<td>37</td>
<td>0x0025</td>
</tr>
<tr>
<td>ampersand</td>
<td>38</td>
<td>0x0026</td>
</tr>
<tr>
<td>quoteright</td>
<td>39</td>
<td>0x0027</td>
</tr>
<tr>
<td>parenleft</td>
<td>40</td>
<td>0x0028</td>
</tr>
<tr>
<td>parenright</td>
<td>41</td>
<td>0x0029</td>
</tr>
<tr>
<td>asterisk</td>
<td>42</td>
<td>0x002a</td>
</tr>
<tr>
<td>plus</td>
<td>43</td>
<td>0x002b</td>
</tr>
<tr>
<td>comma</td>
<td>44</td>
<td>0x002c</td>
</tr>
<tr>
<td>minus</td>
<td>45</td>
<td>0x002d</td>
</tr>
<tr>
<td>period</td>
<td>46</td>
<td>0x002e</td>
</tr>
<tr>
<td>slash</td>
<td>47</td>
<td>0x002f</td>
</tr>
<tr>
<td>0</td>
<td>48</td>
<td>0x0030</td>
</tr>
<tr>
<td>Code</td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x0031</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0x0032</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0x0033</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0x0034</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0x0035</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0x0036</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0x0037</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0x0038</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0x0039</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0x003a</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>0x003b</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>0x003c</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>0x003d</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>0x003e</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0x003f</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>0x0040</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>0x0041</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>0x0042</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>0x0043</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0x0044</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>0x0045</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>0x0046</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>0x0047</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>0x0048</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>0x0049</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>0x004a</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>0x004b</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>0x004c</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>0x004d</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>0x004e</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>0x004f</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>0x0050</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>0x0051</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>0x0052</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>0x0053</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>0x0054</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>0x0055</td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>ASCII Code</td>
<td>Hexadecimal</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>V</td>
<td>86</td>
<td>0x0056</td>
</tr>
<tr>
<td>W</td>
<td>87</td>
<td>0x0057</td>
</tr>
<tr>
<td>X</td>
<td>88</td>
<td>0x0058</td>
</tr>
<tr>
<td>Y</td>
<td>89</td>
<td>0x0059</td>
</tr>
<tr>
<td>Z</td>
<td>90</td>
<td>0x005a</td>
</tr>
<tr>
<td>bracketleft</td>
<td>91</td>
<td>0x005b</td>
</tr>
<tr>
<td>backslash</td>
<td>92</td>
<td>0x005c</td>
</tr>
<tr>
<td>bracketright</td>
<td>93</td>
<td>0x005d</td>
</tr>
<tr>
<td>asciicircum</td>
<td>94</td>
<td>0x005e</td>
</tr>
<tr>
<td>underscore</td>
<td>95</td>
<td>0x005f</td>
</tr>
<tr>
<td>quoteleft</td>
<td>96</td>
<td>0x0060</td>
</tr>
<tr>
<td>a</td>
<td>97</td>
<td>0x0061</td>
</tr>
<tr>
<td>b</td>
<td>98</td>
<td>0x0062</td>
</tr>
<tr>
<td>c</td>
<td>99</td>
<td>0x0063</td>
</tr>
<tr>
<td>d</td>
<td>100</td>
<td>0x0064</td>
</tr>
<tr>
<td>e</td>
<td>101</td>
<td>0x0065</td>
</tr>
<tr>
<td>f</td>
<td>102</td>
<td>0x0066</td>
</tr>
<tr>
<td>g</td>
<td>103</td>
<td>0x0067</td>
</tr>
<tr>
<td>h</td>
<td>104</td>
<td>0x0068</td>
</tr>
<tr>
<td>i</td>
<td>105</td>
<td>0x0069</td>
</tr>
<tr>
<td>j</td>
<td>106</td>
<td>0x006a</td>
</tr>
<tr>
<td>k</td>
<td>107</td>
<td>0x006b</td>
</tr>
<tr>
<td>l</td>
<td>108</td>
<td>0x006c</td>
</tr>
<tr>
<td>m</td>
<td>109</td>
<td>0x006d</td>
</tr>
<tr>
<td>n</td>
<td>110</td>
<td>0x006e</td>
</tr>
<tr>
<td>o</td>
<td>111</td>
<td>0x006f</td>
</tr>
<tr>
<td>p</td>
<td>112</td>
<td>0x0070</td>
</tr>
<tr>
<td>q</td>
<td>113</td>
<td>0x0071</td>
</tr>
<tr>
<td>r</td>
<td>114</td>
<td>0x0072</td>
</tr>
<tr>
<td>s</td>
<td>115</td>
<td>0x0073</td>
</tr>
<tr>
<td>t</td>
<td>116</td>
<td>0x0074</td>
</tr>
<tr>
<td>u</td>
<td>117</td>
<td>0x0075</td>
</tr>
<tr>
<td>v</td>
<td>118</td>
<td>0x0076</td>
</tr>
<tr>
<td>w</td>
<td>119</td>
<td>0x0077</td>
</tr>
<tr>
<td>x</td>
<td>120</td>
<td>0x0078</td>
</tr>
<tr>
<td>y</td>
<td>121</td>
<td>0x0079</td>
</tr>
<tr>
<td>z</td>
<td>122</td>
<td>0x007a</td>
</tr>
<tr>
<td>Character</td>
<td>Code</td>
<td>Hex</td>
</tr>
<tr>
<td>------------------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>braceleft</td>
<td>123</td>
<td>0x007b</td>
</tr>
<tr>
<td>bar</td>
<td>124</td>
<td>0x007c</td>
</tr>
<tr>
<td>braceright</td>
<td>125</td>
<td>0x007d</td>
</tr>
<tr>
<td>asciitilde</td>
<td>126</td>
<td>0x007e</td>
</tr>
<tr>
<td>nobreakspace</td>
<td>160</td>
<td>0x00a0</td>
</tr>
<tr>
<td>exclamationdown</td>
<td>161</td>
<td>0x00a1</td>
</tr>
<tr>
<td>cent</td>
<td>162</td>
<td>0x00a2</td>
</tr>
<tr>
<td>sterling</td>
<td>163</td>
<td>0x00a3</td>
</tr>
<tr>
<td>currency</td>
<td>164</td>
<td>0x00a4</td>
</tr>
<tr>
<td>yen</td>
<td>165</td>
<td>0x00a5</td>
</tr>
<tr>
<td>brokenbar</td>
<td>166</td>
<td>0x00a6</td>
</tr>
<tr>
<td>section</td>
<td>167</td>
<td>0x00a7</td>
</tr>
<tr>
<td>diaeresis</td>
<td>168</td>
<td>0x00a8</td>
</tr>
<tr>
<td>copyright</td>
<td>169</td>
<td>0x00a9</td>
</tr>
<tr>
<td>ordfeminine</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>guillemotleft</td>
<td>171</td>
<td>0x00ab</td>
</tr>
<tr>
<td>notsign</td>
<td>172</td>
<td>0x00ac</td>
</tr>
<tr>
<td>hyphen</td>
<td>173</td>
<td>0x00ad</td>
</tr>
<tr>
<td>registered</td>
<td>174</td>
<td>0x00ae</td>
</tr>
<tr>
<td>macron</td>
<td>175</td>
<td>0x00af</td>
</tr>
<tr>
<td>degree</td>
<td>176</td>
<td>0x00b0</td>
</tr>
<tr>
<td>plusminus</td>
<td>177</td>
<td>0x00b1</td>
</tr>
<tr>
<td>twosuperior</td>
<td>178</td>
<td>0x00b2</td>
</tr>
<tr>
<td>threesuperior</td>
<td>179</td>
<td>0x00b3</td>
</tr>
<tr>
<td>acute</td>
<td>180</td>
<td>0x00b4</td>
</tr>
<tr>
<td>mu</td>
<td>181</td>
<td>0x00b5</td>
</tr>
<tr>
<td>paragraph</td>
<td>182</td>
<td>0x00b6</td>
</tr>
<tr>
<td>periodcentered</td>
<td>183</td>
<td>0x00b7</td>
</tr>
<tr>
<td>cedilla</td>
<td>184</td>
<td>0x00b8</td>
</tr>
<tr>
<td>onesuperior</td>
<td>185</td>
<td>0x00b9</td>
</tr>
<tr>
<td>masculine</td>
<td>186</td>
<td>0x00ba</td>
</tr>
<tr>
<td>guillemotright</td>
<td>187</td>
<td>0x00bb</td>
</tr>
<tr>
<td>onequarter</td>
<td>188</td>
<td>0x00bc</td>
</tr>
<tr>
<td>onehalf</td>
<td>189</td>
<td>0x00bd</td>
</tr>
<tr>
<td>threequarters</td>
<td>190</td>
<td>0x00be</td>
</tr>
<tr>
<td>questiondown</td>
<td>191</td>
<td>0x00bf</td>
</tr>
<tr>
<td>Agrave</td>
<td>192</td>
<td>0x00c0</td>
</tr>
<tr>
<td>Character</td>
<td>Decimal</td>
<td>Hex</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>Aacute</td>
<td>193</td>
<td>0x00c1</td>
</tr>
<tr>
<td>Acircumflex</td>
<td>194</td>
<td>0x00c2</td>
</tr>
<tr>
<td>Atilde</td>
<td>195</td>
<td>0x00c3</td>
</tr>
<tr>
<td>Adiaeresis</td>
<td>196</td>
<td>0x00c4</td>
</tr>
<tr>
<td>Aring</td>
<td>197</td>
<td>0x00c5</td>
</tr>
<tr>
<td>AE</td>
<td>198</td>
<td>0x00c6</td>
</tr>
<tr>
<td>Ccedilla</td>
<td>199</td>
<td>0x00c7</td>
</tr>
<tr>
<td>Egrave</td>
<td>200</td>
<td>0x00c8</td>
</tr>
<tr>
<td>Eacute</td>
<td>201</td>
<td>0x00c9</td>
</tr>
<tr>
<td>Ecircumflex</td>
<td>202</td>
<td>0x00ca</td>
</tr>
<tr>
<td>Ediaeresis</td>
<td>203</td>
<td>0x00cb</td>
</tr>
<tr>
<td>Igrave</td>
<td>204</td>
<td>0x00cc</td>
</tr>
<tr>
<td>Iacute</td>
<td>205</td>
<td>0x00cd</td>
</tr>
<tr>
<td>Icircumflex</td>
<td>206</td>
<td>0x00ce</td>
</tr>
<tr>
<td>Idiaeresis</td>
<td>207</td>
<td>0x00cf</td>
</tr>
<tr>
<td>Eth</td>
<td>208</td>
<td>0x00d0</td>
</tr>
<tr>
<td>Ntilde</td>
<td>209</td>
<td>0x00d1</td>
</tr>
<tr>
<td>Ograve</td>
<td>210</td>
<td>0x00d2</td>
</tr>
<tr>
<td>Oacute</td>
<td>211</td>
<td>0x00d3</td>
</tr>
<tr>
<td>OCircumflex</td>
<td>212</td>
<td>0x00d4</td>
</tr>
<tr>
<td>Otilde</td>
<td>213</td>
<td>0x00d5</td>
</tr>
<tr>
<td>Odiaeresis</td>
<td>214</td>
<td>0x00d6</td>
</tr>
<tr>
<td>Multiply</td>
<td>215</td>
<td>0x00d7</td>
</tr>
<tr>
<td>OOblique</td>
<td>216</td>
<td>0x00d8</td>
</tr>
<tr>
<td>Ugrave</td>
<td>217</td>
<td>0x00d9</td>
</tr>
<tr>
<td>Uacute</td>
<td>218</td>
<td>0x00da</td>
</tr>
<tr>
<td>UCircumflex</td>
<td>219</td>
<td>0x00db</td>
</tr>
<tr>
<td>Udiaeresis</td>
<td>220</td>
<td>0x00dc</td>
</tr>
<tr>
<td>Yacute</td>
<td>221</td>
<td>0x00dd</td>
</tr>
<tr>
<td>Thorn</td>
<td>222</td>
<td>0x00de</td>
</tr>
<tr>
<td>Ssharp</td>
<td>223</td>
<td>0x00df</td>
</tr>
<tr>
<td>Agrave</td>
<td>224</td>
<td>0x00e0</td>
</tr>
<tr>
<td>Aacute</td>
<td>225</td>
<td>0x00e1</td>
</tr>
<tr>
<td>ACircumflex</td>
<td>226</td>
<td>0x00e2</td>
</tr>
<tr>
<td>Atilde</td>
<td>227</td>
<td>0x00e3</td>
</tr>
<tr>
<td>Adiaeresis</td>
<td>228</td>
<td>0x00e4</td>
</tr>
<tr>
<td>Aring</td>
<td>229</td>
<td>0x00e5</td>
</tr>
<tr>
<td>Character</td>
<td>Unicode</td>
<td>Hex</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>ae</td>
<td>230</td>
<td>0x00e6</td>
</tr>
<tr>
<td>ccedilla</td>
<td>231</td>
<td>0x00e7</td>
</tr>
<tr>
<td>egrave</td>
<td>232</td>
<td>0x00e8</td>
</tr>
<tr>
<td>eacute</td>
<td>233</td>
<td>0x00e9</td>
</tr>
<tr>
<td>ecircumflex</td>
<td>234</td>
<td>0x00ea</td>
</tr>
<tr>
<td>ediaeresis</td>
<td>235</td>
<td>0x00eb</td>
</tr>
<tr>
<td>igrave</td>
<td>236</td>
<td>0x00ec</td>
</tr>
<tr>
<td>iacute</td>
<td>237</td>
<td>0x00ed</td>
</tr>
<tr>
<td>icircumflex</td>
<td>238</td>
<td>0x00ee</td>
</tr>
<tr>
<td>idiaeresis</td>
<td>239</td>
<td>0x00ef</td>
</tr>
<tr>
<td>eth</td>
<td>240</td>
<td>0x00f0</td>
</tr>
<tr>
<td>ntilde</td>
<td>241</td>
<td>0x00f1</td>
</tr>
<tr>
<td>ograve</td>
<td>242</td>
<td>0x00f2</td>
</tr>
<tr>
<td>oacute</td>
<td>243</td>
<td>0x00f3</td>
</tr>
<tr>
<td>ocircumflex</td>
<td>244</td>
<td>0x00f4</td>
</tr>
<tr>
<td>otilde</td>
<td>245</td>
<td>0x00f5</td>
</tr>
<tr>
<td>odiaeresis</td>
<td>246</td>
<td>0x00f6</td>
</tr>
<tr>
<td>division</td>
<td>247</td>
<td>0x00f7</td>
</tr>
<tr>
<td>oslash</td>
<td>248</td>
<td>0x00f8</td>
</tr>
<tr>
<td>ugrave</td>
<td>249</td>
<td>0x00f9</td>
</tr>
<tr>
<td>uacute</td>
<td>250</td>
<td>0x00fa</td>
</tr>
<tr>
<td>ucircumflex</td>
<td>251</td>
<td>0x00fb</td>
</tr>
<tr>
<td>udiaeresis</td>
<td>252</td>
<td>0x00fc</td>
</tr>
<tr>
<td>yacute</td>
<td>253</td>
<td>0x00fd</td>
</tr>
<tr>
<td>thorn</td>
<td>254</td>
<td>0x00fe</td>
</tr>
<tr>
<td>ydiaeresis</td>
<td>255</td>
<td>0x00ff</td>
</tr>
<tr>
<td>Aogonek</td>
<td>417</td>
<td>0x01a1</td>
</tr>
<tr>
<td>breve</td>
<td>418</td>
<td>0x01a2</td>
</tr>
<tr>
<td>Lstroke</td>
<td>419</td>
<td>0x01a3</td>
</tr>
<tr>
<td>Lcaron</td>
<td>421</td>
<td>0x01a5</td>
</tr>
<tr>
<td>Sacute</td>
<td>422</td>
<td>0x01a6</td>
</tr>
<tr>
<td>Scaron</td>
<td>425</td>
<td>0x01a9</td>
</tr>
<tr>
<td>Scedilla</td>
<td>426</td>
<td>0x01aa</td>
</tr>
<tr>
<td>Tcaron</td>
<td>427</td>
<td>0x01ab</td>
</tr>
<tr>
<td>Zacute</td>
<td>428</td>
<td>0x01ac</td>
</tr>
<tr>
<td>Character</td>
<td>Code Point</td>
<td>Hex Value</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>Zcaron</td>
<td>430</td>
<td>0x01ae</td>
</tr>
<tr>
<td>Zabovedot</td>
<td>431</td>
<td>0x01af</td>
</tr>
<tr>
<td>aogonek</td>
<td>433</td>
<td>0x01b1</td>
</tr>
<tr>
<td>ogonek</td>
<td>434</td>
<td>0x01b2</td>
</tr>
<tr>
<td>lstroke</td>
<td>435</td>
<td>0x01b3</td>
</tr>
<tr>
<td>lcaron</td>
<td>437</td>
<td>0x01b5</td>
</tr>
<tr>
<td>sacute</td>
<td>438</td>
<td>0x01b6</td>
</tr>
<tr>
<td>caron</td>
<td>439</td>
<td>0x01b7</td>
</tr>
<tr>
<td>scaron</td>
<td>441</td>
<td>0x01b9</td>
</tr>
<tr>
<td>scedilla</td>
<td>442</td>
<td>0x01ba</td>
</tr>
<tr>
<td>tcaron</td>
<td>443</td>
<td>0x01bb</td>
</tr>
<tr>
<td>zacute</td>
<td>444</td>
<td>0x01bc</td>
</tr>
<tr>
<td>doubleacute</td>
<td>445</td>
<td>0x01bd</td>
</tr>
<tr>
<td>zcaron</td>
<td>446</td>
<td>0x01be</td>
</tr>
<tr>
<td>zabovedot</td>
<td>447</td>
<td>0x01bf</td>
</tr>
<tr>
<td>Racute</td>
<td>448</td>
<td>0x01c0</td>
</tr>
<tr>
<td>Abreve</td>
<td>451</td>
<td>0x01c3</td>
</tr>
<tr>
<td>Cacute</td>
<td>454</td>
<td>0x01c6</td>
</tr>
<tr>
<td>Ccaron</td>
<td>456</td>
<td>0x01c8</td>
</tr>
<tr>
<td>Eogonek</td>
<td>458</td>
<td>0x01ca</td>
</tr>
<tr>
<td>Ecaron</td>
<td>460</td>
<td>0x01cc</td>
</tr>
<tr>
<td>Dcaron</td>
<td>463</td>
<td>0x01cf</td>
</tr>
<tr>
<td>Nacute</td>
<td>465</td>
<td>0x01d1</td>
</tr>
<tr>
<td>Ncaron</td>
<td>466</td>
<td>0x01d2</td>
</tr>
<tr>
<td>Odoubleacute</td>
<td>469</td>
<td>0x01d5</td>
</tr>
<tr>
<td>Rcaron</td>
<td>472</td>
<td>0x01d8</td>
</tr>
<tr>
<td>Uring</td>
<td>473</td>
<td>0x01d9</td>
</tr>
<tr>
<td>Udoubleacute</td>
<td>475</td>
<td>0x01db</td>
</tr>
<tr>
<td>Tcedilla</td>
<td>478</td>
<td>0x01de</td>
</tr>
<tr>
<td>racute</td>
<td>480</td>
<td>0x01e0</td>
</tr>
<tr>
<td>abreve</td>
<td>483</td>
<td>0x01e3</td>
</tr>
<tr>
<td>cacute</td>
<td>486</td>
<td>0x01e6</td>
</tr>
<tr>
<td>ccaron</td>
<td>488</td>
<td>0x01e8</td>
</tr>
<tr>
<td>eogonek</td>
<td>490</td>
<td>0x01ea</td>
</tr>
<tr>
<td>ecaron</td>
<td>492</td>
<td>0x01ec</td>
</tr>
<tr>
<td>dcaron</td>
<td>495</td>
<td>0x01ef</td>
</tr>
<tr>
<td>nacute</td>
<td>497</td>
<td>0x01f1</td>
</tr>
<tr>
<td>Character</td>
<td>Dec</td>
<td>Hex</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>ncaron</td>
<td>498</td>
<td>0x01f2</td>
</tr>
<tr>
<td>odoubleacute</td>
<td>501</td>
<td>0x01f5</td>
</tr>
<tr>
<td>rcaron</td>
<td>504</td>
<td>0x01f8</td>
</tr>
<tr>
<td>uring</td>
<td>505</td>
<td>0x01f9</td>
</tr>
<tr>
<td>udoubleacute</td>
<td>507</td>
<td>0x01fb</td>
</tr>
<tr>
<td>tcedilla</td>
<td>510</td>
<td>0x01fe</td>
</tr>
<tr>
<td>abovedot</td>
<td>511</td>
<td>0x01ff</td>
</tr>
<tr>
<td>Hstroke</td>
<td>673</td>
<td>0x02a1</td>
</tr>
<tr>
<td>Hcircumflex</td>
<td>678</td>
<td>0x02a6</td>
</tr>
<tr>
<td>Iabovedot</td>
<td>681</td>
<td>0x02a9</td>
</tr>
<tr>
<td>Gbreve</td>
<td>683</td>
<td>0x02ab</td>
</tr>
<tr>
<td>Jcircumflex</td>
<td>684</td>
<td>0x02ac</td>
</tr>
<tr>
<td>hstroke</td>
<td>689</td>
<td>0x02b1</td>
</tr>
<tr>
<td>hcircumflex</td>
<td>694</td>
<td>0x02b6</td>
</tr>
<tr>
<td>idotless</td>
<td>697</td>
<td>0x02b9</td>
</tr>
<tr>
<td>gbreve</td>
<td>699</td>
<td>0x02bb</td>
</tr>
<tr>
<td>jcircumflex</td>
<td>700</td>
<td>0x02bc</td>
</tr>
<tr>
<td>Cabovedot</td>
<td>709</td>
<td>0x02c5</td>
</tr>
<tr>
<td>Ccircumflex</td>
<td>710</td>
<td>0x02c6</td>
</tr>
<tr>
<td>Gabovedot</td>
<td>725</td>
<td>0x02d5</td>
</tr>
<tr>
<td>Gcircumflex</td>
<td>728</td>
<td>0x02d8</td>
</tr>
<tr>
<td>Ubreve</td>
<td>733</td>
<td>0x02dd</td>
</tr>
<tr>
<td>Scircumflex</td>
<td>734</td>
<td>0x02de</td>
</tr>
<tr>
<td>cabovedot</td>
<td>741</td>
<td>0x02e5</td>
</tr>
<tr>
<td>ccircumflex</td>
<td>742</td>
<td>0x02e6</td>
</tr>
<tr>
<td>gabovedot</td>
<td>757</td>
<td>0x02f5</td>
</tr>
<tr>
<td>gcircumflex</td>
<td>760</td>
<td>0x02f8</td>
</tr>
<tr>
<td>ubreve</td>
<td>765</td>
<td>0x02fd</td>
</tr>
<tr>
<td>scircumflex</td>
<td>766</td>
<td>0x02fe</td>
</tr>
<tr>
<td>kappa</td>
<td>930</td>
<td>0x03a2</td>
</tr>
<tr>
<td>Rcedilla</td>
<td>931</td>
<td>0x03a3</td>
</tr>
<tr>
<td>Itilde</td>
<td>933</td>
<td>0x03a5</td>
</tr>
<tr>
<td>Lcedilla</td>
<td>934</td>
<td>0x03a6</td>
</tr>
<tr>
<td>Emacron</td>
<td>938</td>
<td>0x03aa</td>
</tr>
<tr>
<td>Gcedilla</td>
<td>939</td>
<td>0x03ab</td>
</tr>
<tr>
<td>Tslash</td>
<td>940</td>
<td>0x03ac</td>
</tr>
<tr>
<td>rcedilla</td>
<td>947</td>
<td>0x03b3</td>
</tr>
<tr>
<td>Character</td>
<td>Unicode Codepoint</td>
<td>Hex Code</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>tilde</td>
<td>949</td>
<td>0x03b5</td>
</tr>
<tr>
<td>lcedilla</td>
<td>950</td>
<td>0x03b6</td>
</tr>
<tr>
<td>emacron</td>
<td>954</td>
<td>0x03ba</td>
</tr>
<tr>
<td>gacute</td>
<td>955</td>
<td>0x03bb</td>
</tr>
<tr>
<td>tslash</td>
<td>956</td>
<td>0x03bc</td>
</tr>
<tr>
<td>ENG</td>
<td>957</td>
<td>0x03bd</td>
</tr>
<tr>
<td>eng</td>
<td>959</td>
<td>0x03bf</td>
</tr>
<tr>
<td>Amacron</td>
<td>960</td>
<td>0x03c0</td>
</tr>
<tr>
<td>Iogonek</td>
<td>967</td>
<td>0x03c7</td>
</tr>
<tr>
<td>Eabovedot</td>
<td>972</td>
<td>0x03cc</td>
</tr>
<tr>
<td>Imacron</td>
<td>975</td>
<td>0x03cf</td>
</tr>
<tr>
<td>Ncedilla</td>
<td>977</td>
<td>0x03d1</td>
</tr>
<tr>
<td>Omacron</td>
<td>978</td>
<td>0x03d2</td>
</tr>
<tr>
<td>Kcedilla</td>
<td>979</td>
<td>0x03d3</td>
</tr>
<tr>
<td>Uogonek</td>
<td>985</td>
<td>0x03d9</td>
</tr>
<tr>
<td>Utilde</td>
<td>989</td>
<td>0x03dd</td>
</tr>
<tr>
<td>Umacron</td>
<td>990</td>
<td>0x03de</td>
</tr>
<tr>
<td>amacron</td>
<td>992</td>
<td>0x03e0</td>
</tr>
<tr>
<td>iogonek</td>
<td>999</td>
<td>0x03e7</td>
</tr>
<tr>
<td>eabovedot</td>
<td>1004</td>
<td>0x03ec</td>
</tr>
<tr>
<td>imacron</td>
<td>1007</td>
<td>0x03ef</td>
</tr>
<tr>
<td>ncedilla</td>
<td>1009</td>
<td>0x03f1</td>
</tr>
<tr>
<td>omacron</td>
<td>1010</td>
<td>0x03f2</td>
</tr>
<tr>
<td>kcedilla</td>
<td>1011</td>
<td>0x03f3</td>
</tr>
<tr>
<td>uogonek</td>
<td>1017</td>
<td>0x03f9</td>
</tr>
<tr>
<td>utilde</td>
<td>1021</td>
<td>0x03fd</td>
</tr>
<tr>
<td>umacron</td>
<td>1022</td>
<td>0x03fe</td>
</tr>
<tr>
<td>overline</td>
<td>1150</td>
<td>0x047e</td>
</tr>
<tr>
<td>kana_fullstop</td>
<td>1185</td>
<td>0x04a1</td>
</tr>
<tr>
<td>kana_openingbracket</td>
<td>1186</td>
<td>0x04a2</td>
</tr>
<tr>
<td>kana_closingbracket</td>
<td>1187</td>
<td>0x04a3</td>
</tr>
<tr>
<td>kana_comma</td>
<td>1188</td>
<td>0x04a4</td>
</tr>
<tr>
<td>kana_middledot</td>
<td>1189</td>
<td>0x04a5</td>
</tr>
<tr>
<td>kana_WO</td>
<td>1190</td>
<td>0x04a6</td>
</tr>
<tr>
<td>kana_a</td>
<td>1191</td>
<td>0x04a7</td>
</tr>
<tr>
<td>kana_i</td>
<td>1192</td>
<td>0x04a8</td>
</tr>
<tr>
<td>kana_u</td>
<td>1193</td>
<td>0x04a9</td>
</tr>
<tr>
<td>kana</td>
<td>code</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>1194</td>
<td></td>
</tr>
<tr>
<td>o</td>
<td>1195</td>
<td></td>
</tr>
<tr>
<td>ya</td>
<td>1196</td>
<td></td>
</tr>
<tr>
<td>yu</td>
<td>1197</td>
<td></td>
</tr>
<tr>
<td>yo</td>
<td>1198</td>
<td></td>
</tr>
<tr>
<td>tu</td>
<td>1199</td>
<td></td>
</tr>
<tr>
<td>prolongedsound</td>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1201</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>1202</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>1203</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>1204</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>1205</td>
<td></td>
</tr>
<tr>
<td>KA</td>
<td>1206</td>
<td></td>
</tr>
<tr>
<td>KI</td>
<td>1207</td>
<td></td>
</tr>
<tr>
<td>KU</td>
<td>1208</td>
<td></td>
</tr>
<tr>
<td>KE</td>
<td>1209</td>
<td></td>
</tr>
<tr>
<td>KO</td>
<td>1210</td>
<td></td>
</tr>
<tr>
<td>SA</td>
<td>1211</td>
<td></td>
</tr>
<tr>
<td>SHI</td>
<td>1212</td>
<td></td>
</tr>
<tr>
<td>SU</td>
<td>1213</td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>1214</td>
<td></td>
</tr>
<tr>
<td>S0</td>
<td>1215</td>
<td></td>
</tr>
<tr>
<td>TA</td>
<td>1216</td>
<td></td>
</tr>
<tr>
<td>TI</td>
<td>1217</td>
<td></td>
</tr>
<tr>
<td>TU</td>
<td>1218</td>
<td></td>
</tr>
<tr>
<td>TE</td>
<td>1219</td>
<td></td>
</tr>
<tr>
<td>T0</td>
<td>1220</td>
<td></td>
</tr>
<tr>
<td>NA</td>
<td>1221</td>
<td></td>
</tr>
<tr>
<td>NI</td>
<td>1222</td>
<td></td>
</tr>
<tr>
<td>NU</td>
<td>1223</td>
<td></td>
</tr>
<tr>
<td>NE</td>
<td>1224</td>
<td></td>
</tr>
<tr>
<td>NO</td>
<td>1225</td>
<td></td>
</tr>
<tr>
<td>HA</td>
<td>1226</td>
<td></td>
</tr>
<tr>
<td>HI</td>
<td>1227</td>
<td></td>
</tr>
<tr>
<td>HU</td>
<td>1228</td>
<td></td>
</tr>
<tr>
<td>HE</td>
<td>1229</td>
<td></td>
</tr>
<tr>
<td>HO</td>
<td>1230</td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>Code</td>
<td>Value</td>
</tr>
<tr>
<td>----------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>kana_MA</td>
<td>1231</td>
<td>0x04cf</td>
</tr>
<tr>
<td>kana_MI</td>
<td>1232</td>
<td>0x04d0</td>
</tr>
<tr>
<td>kana_MU</td>
<td>1233</td>
<td>0x04d1</td>
</tr>
<tr>
<td>kana_ME</td>
<td>1234</td>
<td>0x04d2</td>
</tr>
<tr>
<td>kana_MO</td>
<td>1235</td>
<td>0x04d3</td>
</tr>
<tr>
<td>kana_YA</td>
<td>1236</td>
<td>0x04d4</td>
</tr>
<tr>
<td>kana_YU</td>
<td>1237</td>
<td>0x04d5</td>
</tr>
<tr>
<td>kana_YO</td>
<td>1238</td>
<td>0x04d6</td>
</tr>
<tr>
<td>kana_RA</td>
<td>1239</td>
<td>0x04d7</td>
</tr>
<tr>
<td>kana_RI</td>
<td>1240</td>
<td>0x04d8</td>
</tr>
<tr>
<td>kana_RU</td>
<td>1241</td>
<td>0x04d9</td>
</tr>
<tr>
<td>kana_RE</td>
<td>1242</td>
<td>0x04da</td>
</tr>
<tr>
<td>kana_RO</td>
<td>1243</td>
<td>0x04db</td>
</tr>
<tr>
<td>kana_WA</td>
<td>1244</td>
<td>0x04dc</td>
</tr>
<tr>
<td>kana_N</td>
<td>1245</td>
<td>0x04dd</td>
</tr>
<tr>
<td>voicedsound</td>
<td>1246</td>
<td>0x04de</td>
</tr>
<tr>
<td>semivoicedsound</td>
<td>1247</td>
<td>0x04df</td>
</tr>
<tr>
<td>Arabic_comma</td>
<td>1452</td>
<td>0x05ac</td>
</tr>
<tr>
<td>Arabic_semicolon</td>
<td>1467</td>
<td>0x05bb</td>
</tr>
<tr>
<td>Arabic_question_mark</td>
<td>1471</td>
<td>0x05bf</td>
</tr>
<tr>
<td>Arabic_hamza</td>
<td>1473</td>
<td>0x05c1</td>
</tr>
<tr>
<td>Arabic_maddaonalef</td>
<td>1474</td>
<td>0x05c2</td>
</tr>
<tr>
<td>Arabic_hamzaonalef</td>
<td>1475</td>
<td>0x05c3</td>
</tr>
<tr>
<td>Arabic_hamzaonwaw</td>
<td>1476</td>
<td>0x05c4</td>
</tr>
<tr>
<td>Arabic_hamzaounderalef</td>
<td>1477</td>
<td>0x05c5</td>
</tr>
<tr>
<td>Arabic_hamzaonyeh</td>
<td>1478</td>
<td>0x05c6</td>
</tr>
<tr>
<td>Arabic_alef</td>
<td>1479</td>
<td>0x05c7</td>
</tr>
<tr>
<td>Arabic_beh</td>
<td>1480</td>
<td>0x05c8</td>
</tr>
<tr>
<td>Arabic_tehmarbuta</td>
<td>1481</td>
<td>0x05c9</td>
</tr>
<tr>
<td>Arabic_teh</td>
<td>1482</td>
<td>0x05ca</td>
</tr>
<tr>
<td>Arabic_theh</td>
<td>1483</td>
<td>0x05cb</td>
</tr>
<tr>
<td>Arabic_jeem</td>
<td>1484</td>
<td>0x05cc</td>
</tr>
<tr>
<td>Arabic_hah</td>
<td>1485</td>
<td>0x05cd</td>
</tr>
<tr>
<td>Arabic_khah</td>
<td>1486</td>
<td>0x05ce</td>
</tr>
<tr>
<td>Arabic_dal</td>
<td>1487</td>
<td>0x05cf</td>
</tr>
<tr>
<td>Arabic_thal</td>
<td>1488</td>
<td>0x05d0</td>
</tr>
<tr>
<td>Arabic_ra</td>
<td>1489</td>
<td>0x05d1</td>
</tr>
<tr>
<td>Arabic</td>
<td>Code</td>
<td>Unicode</td>
</tr>
<tr>
<td>-------------</td>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>zain</td>
<td>1490</td>
<td>0x05d2</td>
</tr>
<tr>
<td>seen</td>
<td>1491</td>
<td>0x05d3</td>
</tr>
<tr>
<td>sheen</td>
<td>1492</td>
<td>0x05d4</td>
</tr>
<tr>
<td>sad</td>
<td>1493</td>
<td>0x05d5</td>
</tr>
<tr>
<td>dad</td>
<td>1494</td>
<td>0x05d6</td>
</tr>
<tr>
<td>tah</td>
<td>1495</td>
<td>0x05d7</td>
</tr>
<tr>
<td>zah</td>
<td>1496</td>
<td>0x05d8</td>
</tr>
<tr>
<td>ain</td>
<td>1497</td>
<td>0x05d9</td>
</tr>
<tr>
<td>ghain</td>
<td>1498</td>
<td>0x05da</td>
</tr>
<tr>
<td>tatweel</td>
<td>1504</td>
<td>0x05e0</td>
</tr>
<tr>
<td>feh</td>
<td>1505</td>
<td>0x05e1</td>
</tr>
<tr>
<td>qaf</td>
<td>1506</td>
<td>0x05e2</td>
</tr>
<tr>
<td>kaf</td>
<td>1507</td>
<td>0x05e3</td>
</tr>
<tr>
<td>lam</td>
<td>1508</td>
<td>0x05e4</td>
</tr>
<tr>
<td>meem</td>
<td>1509</td>
<td>0x05e5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arabic</th>
<th>Code</th>
<th>Unicode</th>
</tr>
</thead>
<tbody>
<tr>
<td>noon</td>
<td>1510</td>
<td>0x05e6</td>
</tr>
<tr>
<td>heh</td>
<td>1511</td>
<td>0x05e7</td>
</tr>
<tr>
<td>waw</td>
<td>1512</td>
<td>0x05e8</td>
</tr>
<tr>
<td>alefmaksura</td>
<td>1513</td>
<td>0x05e9</td>
</tr>
<tr>
<td>yeh</td>
<td>1514</td>
<td>0x05ea</td>
</tr>
<tr>
<td>fathatan</td>
<td>1515</td>
<td>0x05eb</td>
</tr>
<tr>
<td>dammatan</td>
<td>1516</td>
<td>0x05ec</td>
</tr>
<tr>
<td>kasratan</td>
<td>1517</td>
<td>0x05ed</td>
</tr>
<tr>
<td>fatha</td>
<td>1518</td>
<td>0x05ee</td>
</tr>
<tr>
<td>damma</td>
<td>1519</td>
<td>0x05ef</td>
</tr>
<tr>
<td>kasra</td>
<td>1520</td>
<td>0x05f0</td>
</tr>
<tr>
<td>shadda</td>
<td>1521</td>
<td>0x05f1</td>
</tr>
<tr>
<td>sukun</td>
<td>1522</td>
<td>0x05f2</td>
</tr>
<tr>
<td>dje</td>
<td>1697</td>
<td>0x06a1</td>
</tr>
<tr>
<td>gje</td>
<td>1698</td>
<td>0x06a2</td>
</tr>
<tr>
<td>io</td>
<td>1699</td>
<td>0x06a3</td>
</tr>
<tr>
<td>je</td>
<td>1700</td>
<td>0x06a4</td>
</tr>
<tr>
<td>dse</td>
<td>1701</td>
<td>0x06a5</td>
</tr>
<tr>
<td>i</td>
<td>1702</td>
<td>0x06a6</td>
</tr>
<tr>
<td>Symbol</td>
<td>Code</td>
<td>UTF-8</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>Ukranian_yi</td>
<td>1703</td>
<td>0x06a7</td>
</tr>
<tr>
<td>Serbian_je</td>
<td>1704</td>
<td>0x06a8</td>
</tr>
<tr>
<td>Serbian_lje</td>
<td>1705</td>
<td>0x06a9</td>
</tr>
<tr>
<td>Serbian_nje</td>
<td>1706</td>
<td>0x06aa</td>
</tr>
<tr>
<td>Serbian_tshe</td>
<td>1707</td>
<td>0x06ab</td>
</tr>
<tr>
<td>Macedonia_kje</td>
<td>1708</td>
<td>0x06ac</td>
</tr>
<tr>
<td>Byelorussian_shortu</td>
<td>1710</td>
<td>0x06ae</td>
</tr>
<tr>
<td>Serbian_dze</td>
<td>1711</td>
<td>0x06af</td>
</tr>
<tr>
<td>numerosign</td>
<td>1712</td>
<td>0x06b0</td>
</tr>
<tr>
<td>Serbian_DJE</td>
<td>1713</td>
<td>0x06b1</td>
</tr>
<tr>
<td>Macedonia_GJE</td>
<td>1714</td>
<td>0x06b2</td>
</tr>
<tr>
<td>Cyrillic_IO</td>
<td>1715</td>
<td>0x06b3</td>
</tr>
<tr>
<td>Ukranian_JE</td>
<td>1716</td>
<td>0x06b4</td>
</tr>
<tr>
<td>Macedonia_DSE</td>
<td>1717</td>
<td>0x06b5</td>
</tr>
<tr>
<td>Ukranian_I</td>
<td>1718</td>
<td>0x06b6</td>
</tr>
<tr>
<td>Ukranian_YI</td>
<td>1719</td>
<td>0x06b7</td>
</tr>
<tr>
<td>Serbian_JE</td>
<td>1720</td>
<td>0x06b8</td>
</tr>
<tr>
<td>Serbian_LJE</td>
<td>1721</td>
<td>0x06b9</td>
</tr>
<tr>
<td>Serbian_NJE</td>
<td>1722</td>
<td>0x06ba</td>
</tr>
<tr>
<td>Serbian_TSHE</td>
<td>1723</td>
<td>0x06bb</td>
</tr>
<tr>
<td>Macedonia_KJE</td>
<td>1724</td>
<td>0x06bc</td>
</tr>
<tr>
<td>Byelorussian_SHORTU</td>
<td>1726</td>
<td>0x06be</td>
</tr>
<tr>
<td>Serbian_DZE</td>
<td>1727</td>
<td>0x06bf</td>
</tr>
<tr>
<td>Cyrillic_yu</td>
<td>1728</td>
<td>0x06c0</td>
</tr>
<tr>
<td>Cyrillic_a</td>
<td>1729</td>
<td>0x06c1</td>
</tr>
<tr>
<td>Cyrillic_be</td>
<td>1730</td>
<td>0x06c2</td>
</tr>
<tr>
<td>Cyrillic_tse</td>
<td>1731</td>
<td>0x06c3</td>
</tr>
<tr>
<td>Cyrillic_de</td>
<td>1732</td>
<td>0x06c4</td>
</tr>
<tr>
<td>Cyrillic_ie</td>
<td>1733</td>
<td>0x06c5</td>
</tr>
<tr>
<td>Cyrillic_ef</td>
<td>1734</td>
<td>0x06c6</td>
</tr>
<tr>
<td>Cyrillic_ghe</td>
<td>1735</td>
<td>0x06c7</td>
</tr>
<tr>
<td>Cyrillic_ha</td>
<td>1736</td>
<td>0x06c8</td>
</tr>
<tr>
<td>Cyrillic_i</td>
<td>1737</td>
<td>0x06c9</td>
</tr>
<tr>
<td>Cyrillic_shorti</td>
<td>1738</td>
<td>0x06ca</td>
</tr>
<tr>
<td>Cyrillic_ka</td>
<td>1739</td>
<td>0x06cb</td>
</tr>
<tr>
<td>Cyrillic_el</td>
<td>1740</td>
<td>0x06cc</td>
</tr>
<tr>
<td>Cyrillic_em</td>
<td>1741</td>
<td>0x06cd</td>
</tr>
<tr>
<td>Cyrillic</td>
<td>Code</td>
<td>Value</td>
</tr>
<tr>
<td>----------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>Cyrillic_en</td>
<td>1742</td>
<td>0x06ce</td>
</tr>
<tr>
<td>Cyrillic_o</td>
<td>1743</td>
<td>0x06cf</td>
</tr>
<tr>
<td>Cyrillic_pe</td>
<td>1744</td>
<td>0x06d0</td>
</tr>
<tr>
<td>Cyrillic_ya</td>
<td>1745</td>
<td>0x06d1</td>
</tr>
<tr>
<td>Cyrillic_er</td>
<td>1746</td>
<td>0x06d2</td>
</tr>
<tr>
<td>Cyrillic_es</td>
<td>1747</td>
<td>0x06d3</td>
</tr>
<tr>
<td>Cyrillic_te</td>
<td>1748</td>
<td>0x06d4</td>
</tr>
<tr>
<td>Cyrillic_u</td>
<td>1749</td>
<td>0x06d5</td>
</tr>
<tr>
<td>Cyrillic_zhe</td>
<td>1750</td>
<td>0x06d6</td>
</tr>
<tr>
<td>Cyrillic_ve</td>
<td>1751</td>
<td>0x06d7</td>
</tr>
<tr>
<td>Cyrillic_softsign</td>
<td>1752</td>
<td>0x06d8</td>
</tr>
<tr>
<td>Cyrillic_yeru</td>
<td>1753</td>
<td>0x06d9</td>
</tr>
<tr>
<td>Cyrillic_ze</td>
<td>1754</td>
<td>0x06da</td>
</tr>
<tr>
<td>Cyrillic_sha</td>
<td>1755</td>
<td>0x06db</td>
</tr>
<tr>
<td>Cyrillic_e</td>
<td>1756</td>
<td>0x06dc</td>
</tr>
<tr>
<td>Cyrillic_shcha</td>
<td>1757</td>
<td>0x06dd</td>
</tr>
<tr>
<td>Cyrillic_che</td>
<td>1758</td>
<td>0x06de</td>
</tr>
<tr>
<td>Cyrillic_hardsign</td>
<td>1759</td>
<td>0x06df</td>
</tr>
<tr>
<td>Cyrillic_YU</td>
<td>1760</td>
<td>0x06e0</td>
</tr>
<tr>
<td>Cyrillic_A</td>
<td>1761</td>
<td>0x06e1</td>
</tr>
<tr>
<td>Cyrillic_BE</td>
<td>1762</td>
<td>0x06e2</td>
</tr>
<tr>
<td>Cyrillic_TSE</td>
<td>1763</td>
<td>0x06e3</td>
</tr>
<tr>
<td>Cyrillic_DE</td>
<td>1764</td>
<td>0x06e4</td>
</tr>
<tr>
<td>Cyrillic_IE</td>
<td>1765</td>
<td>0x06e5</td>
</tr>
<tr>
<td>Cyrillic_EF</td>
<td>1766</td>
<td>0x06e6</td>
</tr>
<tr>
<td>Cyrillic_GHE</td>
<td>1767</td>
<td>0x06e7</td>
</tr>
<tr>
<td>Cyrillic_HA</td>
<td>1768</td>
<td>0x06e8</td>
</tr>
<tr>
<td>Cyrillic_I</td>
<td>1769</td>
<td>0x06e9</td>
</tr>
<tr>
<td>Cyrillic_SHORTI</td>
<td>1770</td>
<td>0x06ea</td>
</tr>
<tr>
<td>Cyrillic_KA</td>
<td>1771</td>
<td>0x06eb</td>
</tr>
<tr>
<td>Cyrillic_EL</td>
<td>1772</td>
<td>0x06ec</td>
</tr>
<tr>
<td>Cyrillic_EM</td>
<td>1773</td>
<td>0x06ed</td>
</tr>
<tr>
<td>Cyrillic_EN</td>
<td>1774</td>
<td>0x06ee</td>
</tr>
<tr>
<td>Cyrillic_O</td>
<td>1775</td>
<td>0x06ef</td>
</tr>
<tr>
<td>Cyrillic_PE</td>
<td>1776</td>
<td>0x06f0</td>
</tr>
<tr>
<td>Cyrillic_YA</td>
<td>1777</td>
<td>0x06f1</td>
</tr>
<tr>
<td>Cyrillic_ER</td>
<td>1778</td>
<td>0x06f2</td>
</tr>
<tr>
<td>Character</td>
<td>Code</td>
<td>Number</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>Cyrillic_ES</td>
<td>1779</td>
<td>0x06f3</td>
</tr>
<tr>
<td>Cyrillic_TE</td>
<td>1780</td>
<td>0x06f4</td>
</tr>
<tr>
<td>Cyrillic_U</td>
<td>1781</td>
<td>0x06f5</td>
</tr>
<tr>
<td>Cyrillic_ZHE</td>
<td>1782</td>
<td>0x06f6</td>
</tr>
<tr>
<td>Cyrillic_VE</td>
<td>1783</td>
<td>0x06f7</td>
</tr>
<tr>
<td>Cyrillic_SOFTSIGN</td>
<td>1784</td>
<td>0x06f8</td>
</tr>
<tr>
<td>Cyrillic_YERU</td>
<td>1785</td>
<td>0x06f9</td>
</tr>
<tr>
<td>Cyrillic_ZE</td>
<td>1786</td>
<td>0x06fa</td>
</tr>
<tr>
<td>Cyrillic_SHA</td>
<td>1787</td>
<td>0x06fb</td>
</tr>
<tr>
<td>Cyrillic_E</td>
<td>1788</td>
<td>0x06fc</td>
</tr>
<tr>
<td>Cyrillic_SHCHA</td>
<td>1789</td>
<td>0x06fd</td>
</tr>
<tr>
<td>Cyrillic_CHE</td>
<td>1790</td>
<td>0x06fe</td>
</tr>
<tr>
<td>Cyrillic_HARDSIGN</td>
<td>1791</td>
<td>0x06ff</td>
</tr>
<tr>
<td>Greek_ALPHAaccent</td>
<td>1953</td>
<td>0x07a1</td>
</tr>
<tr>
<td>Greek_EPSILONaccent</td>
<td>1954</td>
<td>0x07a2</td>
</tr>
<tr>
<td>Greek_ETAaccent</td>
<td>1955</td>
<td>0x07a3</td>
</tr>
<tr>
<td>Greek_IOTAaccent</td>
<td>1956</td>
<td>0x07a4</td>
</tr>
<tr>
<td>Greek_IOTAaccentdiaeresis</td>
<td>1957</td>
<td>0x07a5</td>
</tr>
<tr>
<td>Greek_IOTAaccentdieresis</td>
<td>1958</td>
<td>0x07a6</td>
</tr>
<tr>
<td>Greek_OMICRONaccent</td>
<td>1959</td>
<td>0x07a7</td>
</tr>
<tr>
<td>Greek_UPSILONaccent</td>
<td>1960</td>
<td>0x07a8</td>
</tr>
<tr>
<td>Greek_UPSILONdieresis</td>
<td>1961</td>
<td>0x07a9</td>
</tr>
<tr>
<td>Greek_UPSILONaccentdieresis</td>
<td>1962</td>
<td>0x07aa</td>
</tr>
<tr>
<td>Greek_OMEGAaccent</td>
<td>1963</td>
<td>0x07ab</td>
</tr>
<tr>
<td>Greek_alphaaccent</td>
<td>1969</td>
<td>0x07b1</td>
</tr>
<tr>
<td>Greek_epsilonaccent</td>
<td>1970</td>
<td>0x07b2</td>
</tr>
<tr>
<td>Greek_etalaccent</td>
<td>1971</td>
<td>0x07b3</td>
</tr>
<tr>
<td>Greek_iotaaccent</td>
<td>1972</td>
<td>0x07b4</td>
</tr>
<tr>
<td>Greek_iotadieresis</td>
<td>1973</td>
<td>0x07b5</td>
</tr>
<tr>
<td>Greek_iotaaccentdieresis</td>
<td>1974</td>
<td>0x07b6</td>
</tr>
<tr>
<td>Greek_omicronaccent</td>
<td>1975</td>
<td>0x07b7</td>
</tr>
<tr>
<td>Greek_upsilonaccent</td>
<td>1976</td>
<td>0x07b8</td>
</tr>
<tr>
<td>Greek_upsilondieresis</td>
<td>1977</td>
<td>0x07b9</td>
</tr>
<tr>
<td>Greek_upsilonaccentdieresis</td>
<td>1978</td>
<td>0x07ba</td>
</tr>
<tr>
<td>Greek_omegacent</td>
<td>1979</td>
<td>0x07bb</td>
</tr>
<tr>
<td>Greek_ALPHA</td>
<td>1985</td>
<td>0x07c1</td>
</tr>
<tr>
<td>Greek_BETA</td>
<td>1986</td>
<td>0x07c2</td>
</tr>
<tr>
<td>Greek</td>
<td>Year</td>
<td>Code</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>GAMMA</td>
<td>1987</td>
<td>0x07c3</td>
</tr>
<tr>
<td>DELTA</td>
<td>1988</td>
<td>0x07c4</td>
</tr>
<tr>
<td>EPSILON</td>
<td>1989</td>
<td>0x07c5</td>
</tr>
<tr>
<td>ZETA</td>
<td>1990</td>
<td>0x07c6</td>
</tr>
<tr>
<td>ETA</td>
<td>1991</td>
<td>0x07c7</td>
</tr>
<tr>
<td>THETA</td>
<td>1992</td>
<td>0x07c8</td>
</tr>
<tr>
<td>IOTA</td>
<td>1993</td>
<td>0x07c9</td>
</tr>
<tr>
<td>KAPPA</td>
<td>1994</td>
<td>0x07ca</td>
</tr>
<tr>
<td>LAMBDA</td>
<td>1995</td>
<td>0x07cb</td>
</tr>
<tr>
<td>MU</td>
<td>1996</td>
<td>0x07cc</td>
</tr>
<tr>
<td>NU</td>
<td>1997</td>
<td>0x07cd</td>
</tr>
<tr>
<td>XI</td>
<td>1998</td>
<td>0x07ce</td>
</tr>
<tr>
<td>OMICRON</td>
<td>1999</td>
<td>0x07cf</td>
</tr>
<tr>
<td>PI</td>
<td>2000</td>
<td>0x07d0</td>
</tr>
<tr>
<td>RHO</td>
<td>2001</td>
<td>0x07d1</td>
</tr>
<tr>
<td>SIGMA</td>
<td>2002</td>
<td>0x07d2</td>
</tr>
<tr>
<td>TAU</td>
<td>2004</td>
<td>0x07d4</td>
</tr>
<tr>
<td>UPSILON</td>
<td>2005</td>
<td>0x07d5</td>
</tr>
<tr>
<td>PHI</td>
<td>2006</td>
<td>0x07d6</td>
</tr>
<tr>
<td>CHI</td>
<td>2007</td>
<td>0x07d7</td>
</tr>
<tr>
<td>PSI</td>
<td>2008</td>
<td>0x07d8</td>
</tr>
<tr>
<td>OMEGA</td>
<td>2009</td>
<td>0x07d9</td>
</tr>
<tr>
<td>alpha</td>
<td>2017</td>
<td>0x07e1</td>
</tr>
<tr>
<td>beta</td>
<td>2018</td>
<td>0x07e2</td>
</tr>
<tr>
<td>gamma</td>
<td>2019</td>
<td>0x07e3</td>
</tr>
<tr>
<td>delta</td>
<td>2020</td>
<td>0x07e4</td>
</tr>
<tr>
<td>epsilon</td>
<td>2021</td>
<td>0x07e5</td>
</tr>
<tr>
<td>zeta</td>
<td>2022</td>
<td>0x07e6</td>
</tr>
<tr>
<td>eta</td>
<td>2023</td>
<td>0x07e7</td>
</tr>
<tr>
<td>theta</td>
<td>2024</td>
<td>0x07e8</td>
</tr>
<tr>
<td>iota</td>
<td>2025</td>
<td>0x07e9</td>
</tr>
<tr>
<td>kappa</td>
<td>2026</td>
<td>0x07ea</td>
</tr>
<tr>
<td>lambda</td>
<td>2027</td>
<td>0x07eb</td>
</tr>
<tr>
<td>mu</td>
<td>2028</td>
<td>0x07ec</td>
</tr>
<tr>
<td>nu</td>
<td>2029</td>
<td>0x07ed</td>
</tr>
<tr>
<td>xi</td>
<td>2030</td>
<td>0x07ee</td>
</tr>
<tr>
<td>omicron</td>
<td>2031</td>
<td>0x07ef</td>
</tr>
<tr>
<td>Character</td>
<td>Code</td>
<td>Unicode</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>Greek_pi</td>
<td>2032</td>
<td>0x07f0</td>
</tr>
<tr>
<td>Greek_rho</td>
<td>2033</td>
<td>0x07f1</td>
</tr>
<tr>
<td>Greek_sigma</td>
<td>2034</td>
<td>0x07f2</td>
</tr>
<tr>
<td>Greek_finalsmallsigma</td>
<td>2035</td>
<td>0x07f3</td>
</tr>
<tr>
<td>Greek_tau</td>
<td>2036</td>
<td>0x07f4</td>
</tr>
<tr>
<td>Greek_upsilon</td>
<td>2037</td>
<td>0x07f5</td>
</tr>
<tr>
<td>Greek_phi</td>
<td>2038</td>
<td>0x07f6</td>
</tr>
<tr>
<td>Greek_chi</td>
<td>2039</td>
<td>0x07f7</td>
</tr>
<tr>
<td>Greek_psi</td>
<td>2040</td>
<td>0x07f8</td>
</tr>
<tr>
<td>Greek_omega</td>
<td>2041</td>
<td>0x07f9</td>
</tr>
<tr>
<td>leftradical</td>
<td>2209</td>
<td>0x08a1</td>
</tr>
<tr>
<td>topleftradical</td>
<td>2210</td>
<td>0x08a2</td>
</tr>
<tr>
<td>horizconnector</td>
<td>2211</td>
<td>0x08a3</td>
</tr>
<tr>
<td>topintegral</td>
<td>2212</td>
<td>0x08a4</td>
</tr>
<tr>
<td>botintegral</td>
<td>2213</td>
<td>0x08a5</td>
</tr>
<tr>
<td>vertconnector</td>
<td>2214</td>
<td>0x08a6</td>
</tr>
<tr>
<td>topleftsqbracket</td>
<td>2215</td>
<td>0x08a7</td>
</tr>
<tr>
<td>botleftsqbracket</td>
<td>2216</td>
<td>0x08a8</td>
</tr>
<tr>
<td>toprightsqbracket</td>
<td>2217</td>
<td>0x08a9</td>
</tr>
<tr>
<td>botrightsqbracket</td>
<td>2218</td>
<td>0x08aa</td>
</tr>
<tr>
<td>topleftparens</td>
<td>2219</td>
<td>0x08ab</td>
</tr>
<tr>
<td>botleftparens</td>
<td>2220</td>
<td>0x08ac</td>
</tr>
<tr>
<td>toprightparens</td>
<td>2221</td>
<td>0x08ad</td>
</tr>
<tr>
<td>botrightparens</td>
<td>2222</td>
<td>0x08ae</td>
</tr>
<tr>
<td>leftmiddlecurlybrace</td>
<td>2223</td>
<td>0x08af</td>
</tr>
<tr>
<td>rightmiddlecurlybrace</td>
<td>2224</td>
<td>0x08b0</td>
</tr>
<tr>
<td>topleftsummation</td>
<td>2225</td>
<td>0x08b1</td>
</tr>
<tr>
<td>botleftsummation</td>
<td>2226</td>
<td>0x08b2</td>
</tr>
<tr>
<td>topvertsummationconnector</td>
<td>2227</td>
<td>0x08b3</td>
</tr>
<tr>
<td>botvertsummationconnector</td>
<td>2228</td>
<td>0x08b4</td>
</tr>
<tr>
<td>toprightsummation</td>
<td>2229</td>
<td>0x08b5</td>
</tr>
<tr>
<td>botrightsummation</td>
<td>2230</td>
<td>0x08b6</td>
</tr>
<tr>
<td>rightmiddlesummation</td>
<td>2231</td>
<td>0x08b7</td>
</tr>
<tr>
<td>lesssthanequal</td>
<td>2236</td>
<td>0x08bc</td>
</tr>
<tr>
<td>Symbol</td>
<td>Code</td>
<td>Hex</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>notequal</td>
<td>2237</td>
<td>0x08bd</td>
</tr>
<tr>
<td>greaterthanequal</td>
<td>2238</td>
<td>0x08be</td>
</tr>
<tr>
<td>integral</td>
<td>2239</td>
<td>0x08bf</td>
</tr>
<tr>
<td>therefore</td>
<td>2240</td>
<td>0x08c0</td>
</tr>
<tr>
<td>variation</td>
<td>2241</td>
<td>0x08c1</td>
</tr>
<tr>
<td>infinity</td>
<td>2242</td>
<td>0x08c2</td>
</tr>
<tr>
<td>nabla</td>
<td>2245</td>
<td>0x08c5</td>
</tr>
<tr>
<td>approximate</td>
<td>2248</td>
<td>0x08c8</td>
</tr>
<tr>
<td>similarequal</td>
<td>2249</td>
<td>0x08c9</td>
</tr>
<tr>
<td>ifonlyif</td>
<td>2253</td>
<td>0x08cd</td>
</tr>
<tr>
<td>implies</td>
<td>2254</td>
<td>0x08ce</td>
</tr>
<tr>
<td>identical</td>
<td>2255</td>
<td>0x08cf</td>
</tr>
<tr>
<td>radical</td>
<td>2262</td>
<td>0x08d6</td>
</tr>
<tr>
<td>includedin</td>
<td>2266</td>
<td>0x08da</td>
</tr>
<tr>
<td>includes</td>
<td>2267</td>
<td>0x08db</td>
</tr>
<tr>
<td>intersection</td>
<td>2268</td>
<td>0x08dc</td>
</tr>
<tr>
<td>union</td>
<td>2269</td>
<td>0x08dd</td>
</tr>
<tr>
<td>logicaland</td>
<td>2270</td>
<td>0x08de</td>
</tr>
<tr>
<td>logicalor</td>
<td>2271</td>
<td>0x08df</td>
</tr>
<tr>
<td>partialderivative</td>
<td>2287</td>
<td>0x08ef</td>
</tr>
<tr>
<td>function</td>
<td>2294</td>
<td>0x08f6</td>
</tr>
<tr>
<td>leftarrow</td>
<td>2299</td>
<td>0x08fb</td>
</tr>
<tr>
<td>uparrow</td>
<td>2300</td>
<td>0x08fc</td>
</tr>
<tr>
<td>rightarrow</td>
<td>2301</td>
<td>0x08fd</td>
</tr>
<tr>
<td>downarrow</td>
<td>2302</td>
<td>0x08fe</td>
</tr>
<tr>
<td>blank</td>
<td>2527</td>
<td>0x09df</td>
</tr>
<tr>
<td>soliddiamond</td>
<td>2528</td>
<td>0x09e0</td>
</tr>
<tr>
<td>checkerboard</td>
<td>2529</td>
<td>0x09e1</td>
</tr>
<tr>
<td>ht</td>
<td>2530</td>
<td>0x09e2</td>
</tr>
<tr>
<td>ff</td>
<td>2531</td>
<td>0x09e3</td>
</tr>
<tr>
<td>cr</td>
<td>2532</td>
<td>0x09e4</td>
</tr>
<tr>
<td>lf</td>
<td>2533</td>
<td>0x09e5</td>
</tr>
<tr>
<td>nl</td>
<td>2536</td>
<td>0x09e8</td>
</tr>
<tr>
<td>vt</td>
<td>2537</td>
<td>0x09e9</td>
</tr>
<tr>
<td>lowrightcorner</td>
<td>2538</td>
<td>0x09ea</td>
</tr>
<tr>
<td>uprightcorner</td>
<td>2539</td>
<td>0x09eb</td>
</tr>
<tr>
<td>upleftcorner</td>
<td>2540</td>
<td>0x09ec</td>
</tr>
<tr>
<td>Symbol</td>
<td>Unicode</td>
<td>Hex</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>rightanglebracket</td>
<td>2750</td>
<td>0x0abe</td>
</tr>
<tr>
<td>marker</td>
<td>2751</td>
<td>0x0abf</td>
</tr>
<tr>
<td>oneeighth</td>
<td>2755</td>
<td>0x0ac3</td>
</tr>
<tr>
<td>threeeighths</td>
<td>2756</td>
<td>0x0ac4</td>
</tr>
<tr>
<td>fiveeighths</td>
<td>2757</td>
<td>0x0ac5</td>
</tr>
<tr>
<td>seveneighths</td>
<td>2758</td>
<td>0x0ac6</td>
</tr>
<tr>
<td>trademark</td>
<td>2761</td>
<td>0x0ac9</td>
</tr>
<tr>
<td>signaturemark</td>
<td>2762</td>
<td>0x0aca</td>
</tr>
<tr>
<td>trademarkincircle</td>
<td>2763</td>
<td>0x0acb</td>
</tr>
<tr>
<td>leftopentriangle</td>
<td>2764</td>
<td>0x0acc</td>
</tr>
<tr>
<td>rightopentriangle</td>
<td>2765</td>
<td>0x0acd</td>
</tr>
<tr>
<td>emopencircle</td>
<td>2766</td>
<td>0x0ace</td>
</tr>
<tr>
<td>emopenrectangle</td>
<td>2767</td>
<td>0x0acf</td>
</tr>
<tr>
<td>leftsinglequotemark</td>
<td>2768</td>
<td>0x0ad0</td>
</tr>
<tr>
<td>rightsinglequotemark</td>
<td>2769</td>
<td>0x0ad1</td>
</tr>
<tr>
<td>leftdoublequotemark</td>
<td>2770</td>
<td>0x0ad2</td>
</tr>
<tr>
<td>rightdoublequotemark</td>
<td>2771</td>
<td>0x0ad3</td>
</tr>
<tr>
<td>prescription</td>
<td>2772</td>
<td>0x0ad4</td>
</tr>
<tr>
<td>minutes</td>
<td>2774</td>
<td>0x0ad6</td>
</tr>
<tr>
<td>seconds</td>
<td>2775</td>
<td>0x0ad7</td>
</tr>
<tr>
<td>latincross</td>
<td>2777</td>
<td>0x0ad9</td>
</tr>
<tr>
<td>hexagram</td>
<td>2778</td>
<td>0x0ada</td>
</tr>
<tr>
<td>filledrectbullet</td>
<td>2779</td>
<td>0x0adb</td>
</tr>
<tr>
<td>filledlefttribullet</td>
<td>2780</td>
<td>0x0adc</td>
</tr>
<tr>
<td>filledrighttribullet</td>
<td>2781</td>
<td>0x0add</td>
</tr>
<tr>
<td>emfilledcircbullet</td>
<td>2782</td>
<td>0x0ade</td>
</tr>
<tr>
<td>emfilledrect</td>
<td>2783</td>
<td>0x0adf</td>
</tr>
<tr>
<td>enopencircbullet</td>
<td>2784</td>
<td>0x0ae0</td>
</tr>
<tr>
<td>enopensquarebullet</td>
<td>2785</td>
<td>0x0ae1</td>
</tr>
<tr>
<td>openrectbullet</td>
<td>2786</td>
<td>0x0ae2</td>
</tr>
<tr>
<td>opentrribullet</td>
<td>2787</td>
<td>0x0ae3</td>
</tr>
<tr>
<td>opentrribulletdown</td>
<td>2788</td>
<td>0x0ae4</td>
</tr>
<tr>
<td>openstar</td>
<td>2789</td>
<td>0x0ae5</td>
</tr>
<tr>
<td>enfilledcircbullet</td>
<td>2790</td>
<td>0x0ae6</td>
</tr>
<tr>
<td>enfilledsqbullet</td>
<td>2791</td>
<td>0x0ae7</td>
</tr>
<tr>
<td>filledtribulletup</td>
<td>2792</td>
<td>0x0ae8</td>
</tr>
<tr>
<td>filledtribulletdown</td>
<td>2793</td>
<td>0x0ae9</td>
</tr>
<tr>
<td>Symbol</td>
<td>Code</td>
<td></td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>leftpointer</td>
<td>2794</td>
<td></td>
</tr>
<tr>
<td>rightpointer</td>
<td>2795</td>
<td></td>
</tr>
<tr>
<td>club</td>
<td>2796</td>
<td></td>
</tr>
<tr>
<td>diamond</td>
<td>2797</td>
<td></td>
</tr>
<tr>
<td>heart</td>
<td>2798</td>
<td></td>
</tr>
<tr>
<td>maltesecross</td>
<td>2800</td>
<td></td>
</tr>
<tr>
<td>dagger</td>
<td>2801</td>
<td></td>
</tr>
<tr>
<td>doubledagger</td>
<td>2802</td>
<td></td>
</tr>
<tr>
<td>checkmark</td>
<td>2803</td>
<td></td>
</tr>
<tr>
<td>ballotcross</td>
<td>2804</td>
<td></td>
</tr>
<tr>
<td>musicalsharp</td>
<td>2805</td>
<td></td>
</tr>
<tr>
<td>musicalflat</td>
<td>2806</td>
<td></td>
</tr>
<tr>
<td>malesymbol</td>
<td>2807</td>
<td></td>
</tr>
<tr>
<td>femalesymbol</td>
<td>2808</td>
<td></td>
</tr>
<tr>
<td>telephone</td>
<td>2809</td>
<td></td>
</tr>
<tr>
<td>phonographcopyright</td>
<td>2810</td>
<td></td>
</tr>
<tr>
<td>caret</td>
<td>2811</td>
<td></td>
</tr>
<tr>
<td>singlelowquotemark</td>
<td>2812</td>
<td></td>
</tr>
<tr>
<td>doublelowquotemark</td>
<td>2813</td>
<td></td>
</tr>
<tr>
<td>cursor</td>
<td>2814</td>
<td></td>
</tr>
<tr>
<td>leftcaret</td>
<td>2815</td>
<td></td>
</tr>
<tr>
<td>rightcaret</td>
<td>2816</td>
<td></td>
</tr>
<tr>
<td>downcaret</td>
<td>2817</td>
<td></td>
</tr>
<tr>
<td>upcaret</td>
<td>2818</td>
<td></td>
</tr>
<tr>
<td>overbar</td>
<td>2979</td>
<td></td>
</tr>
<tr>
<td>downtack</td>
<td>2982</td>
<td></td>
</tr>
<tr>
<td>upshoe</td>
<td>2984</td>
<td></td>
</tr>
<tr>
<td>upstile</td>
<td>2985</td>
<td></td>
</tr>
<tr>
<td>underbar</td>
<td>3008</td>
<td></td>
</tr>
<tr>
<td>quad</td>
<td>3010</td>
<td></td>
</tr>
<tr>
<td>uptack</td>
<td>3011</td>
<td></td>
</tr>
<tr>
<td>downstile</td>
<td>3012</td>
<td></td>
</tr>
<tr>
<td>underbar</td>
<td>3013</td>
<td></td>
</tr>
<tr>
<td>jot</td>
<td>3014</td>
<td></td>
</tr>
<tr>
<td>quad</td>
<td>3018</td>
<td></td>
</tr>
<tr>
<td>uptack</td>
<td>3020</td>
<td></td>
</tr>
<tr>
<td>circle</td>
<td>3022</td>
<td></td>
</tr>
<tr>
<td>upstile</td>
<td>3023</td>
<td></td>
</tr>
<tr>
<td>downshoe</td>
<td>3027</td>
<td></td>
</tr>
<tr>
<td>rightshoe</td>
<td>3030</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>Code</td>
<td>ASCII</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>leftshoe</td>
<td>3034</td>
<td>0x0bda</td>
</tr>
<tr>
<td>lefttack</td>
<td>3036</td>
<td>0x0bdc</td>
</tr>
<tr>
<td>righttack</td>
<td>3068</td>
<td>0x0bfc</td>
</tr>
<tr>
<td>hebrew_aleph</td>
<td>3296</td>
<td>0x0ce0</td>
</tr>
<tr>
<td>hebrew_beth</td>
<td>3297</td>
<td>0x0ce1</td>
</tr>
<tr>
<td>hebrew_gimmel</td>
<td>3298</td>
<td>0x0ce2</td>
</tr>
<tr>
<td>hebrew_daleth</td>
<td>3299</td>
<td>0x0ce3</td>
</tr>
<tr>
<td>hebrew_he</td>
<td>3300</td>
<td>0x0ce4</td>
</tr>
<tr>
<td>hebrew_waw</td>
<td>3301</td>
<td>0x0ce5</td>
</tr>
<tr>
<td>hebrew_zayin</td>
<td>3302</td>
<td>0x0ce6</td>
</tr>
<tr>
<td>hebrew_het</td>
<td>3303</td>
<td>0x0ce7</td>
</tr>
<tr>
<td>hebrew_teth</td>
<td>3304</td>
<td>0x0ce8</td>
</tr>
<tr>
<td>hebrew_yod</td>
<td>3305</td>
<td>0x0ce9</td>
</tr>
<tr>
<td>hebrew_finalkaph</td>
<td>3306</td>
<td>0x0cea</td>
</tr>
<tr>
<td>hebrew_kaph</td>
<td>3307</td>
<td>0x0ceb</td>
</tr>
<tr>
<td>hebrew_lamed</td>
<td>3308</td>
<td>0x0cec</td>
</tr>
<tr>
<td>hebrew_finalmem</td>
<td>3309</td>
<td>0x0ced</td>
</tr>
<tr>
<td>hebrew_mem</td>
<td>3310</td>
<td>0x0cee</td>
</tr>
<tr>
<td>hebrew_finalnun</td>
<td>3311</td>
<td>0x0cef</td>
</tr>
<tr>
<td>hebrew_nun</td>
<td>3312</td>
<td>0x0cf0</td>
</tr>
<tr>
<td>hebrew_samekh</td>
<td>3313</td>
<td>0x0cf1</td>
</tr>
<tr>
<td>hebrew_ayin</td>
<td>3314</td>
<td>0x0cf2</td>
</tr>
<tr>
<td>hebrew_finalpe</td>
<td>3315</td>
<td>0x0cf3</td>
</tr>
<tr>
<td>hebrew_pe</td>
<td>3316</td>
<td>0x0cf4</td>
</tr>
<tr>
<td>hebrew_finalzadi</td>
<td>3317</td>
<td>0x0cf5</td>
</tr>
<tr>
<td>hebrew_zadi</td>
<td>3318</td>
<td>0x0cf6</td>
</tr>
<tr>
<td>hebrew_kuf</td>
<td>3319</td>
<td>0x0cf7</td>
</tr>
<tr>
<td>hebrew_resh</td>
<td>3320</td>
<td>0x0cf8</td>
</tr>
<tr>
<td>hebrew_shin</td>
<td>3321</td>
<td>0x0cf9</td>
</tr>
<tr>
<td>hebrew_taf</td>
<td>3322</td>
<td>0x0cfa</td>
</tr>
<tr>
<td>BackSpace</td>
<td>65288</td>
<td>0xff08</td>
</tr>
<tr>
<td>Tab</td>
<td>65289</td>
<td>0xff09</td>
</tr>
<tr>
<td>Linefeed</td>
<td>65290</td>
<td>0xff0a</td>
</tr>
<tr>
<td>Clear</td>
<td>65291</td>
<td>0xff0b</td>
</tr>
<tr>
<td>Return</td>
<td>65293</td>
<td>0xff0d</td>
</tr>
<tr>
<td>Pause</td>
<td>65299</td>
<td>0xff13</td>
</tr>
<tr>
<td>Scroll_Lock</td>
<td>65300</td>
<td>0xff14</td>
</tr>
<tr>
<td>Key</td>
<td>Code</td>
<td>ASCII</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>Sys_Req</td>
<td>65301</td>
<td>0xff15</td>
</tr>
<tr>
<td>Escape</td>
<td>65307</td>
<td>0xff1b</td>
</tr>
<tr>
<td>Multi_key</td>
<td>65312</td>
<td>0xff20</td>
</tr>
<tr>
<td>Kanji</td>
<td>65313</td>
<td>0xff21</td>
</tr>
<tr>
<td>Home</td>
<td>65360</td>
<td>0xff50</td>
</tr>
<tr>
<td>Left</td>
<td>65361</td>
<td>0xff51</td>
</tr>
<tr>
<td>Up</td>
<td>65362</td>
<td>0xff52</td>
</tr>
<tr>
<td>Right</td>
<td>65363</td>
<td>0xff53</td>
</tr>
<tr>
<td>Down</td>
<td>65364</td>
<td>0xff54</td>
</tr>
<tr>
<td>Prior</td>
<td>65365</td>
<td>0xff55</td>
</tr>
<tr>
<td>Next</td>
<td>65366</td>
<td>0xff56</td>
</tr>
<tr>
<td>End</td>
<td>65367</td>
<td>0xff57</td>
</tr>
<tr>
<td>Begin</td>
<td>65368</td>
<td>0xff58</td>
</tr>
<tr>
<td>Win_L</td>
<td>65371</td>
<td>0xff5b</td>
</tr>
<tr>
<td>Win_R</td>
<td>65372</td>
<td>0xff5c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Code</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>App</td>
<td>65373</td>
<td>0xff5d</td>
</tr>
<tr>
<td>Select</td>
<td>65376</td>
<td>0xff60</td>
</tr>
<tr>
<td>Print</td>
<td>65377</td>
<td>0xff61</td>
</tr>
<tr>
<td>Execute</td>
<td>65378</td>
<td>0xff62</td>
</tr>
<tr>
<td>Insert</td>
<td>65379</td>
<td>0xff63</td>
</tr>
<tr>
<td>Undo</td>
<td>65381</td>
<td>0xff65</td>
</tr>
<tr>
<td>Redo</td>
<td>65382</td>
<td>0xff66</td>
</tr>
<tr>
<td>Menu</td>
<td>65383</td>
<td>0xff67</td>
</tr>
<tr>
<td>Find</td>
<td>65384</td>
<td>0xff68</td>
</tr>
<tr>
<td>Cancel</td>
<td>65385</td>
<td>0xff69</td>
</tr>
<tr>
<td>Help</td>
<td>65386</td>
<td>0xff6a</td>
</tr>
<tr>
<td>Break</td>
<td>65387</td>
<td>0xff6b</td>
</tr>
<tr>
<td>Hebrew_switch</td>
<td>65406</td>
<td>0xff7e</td>
</tr>
<tr>
<td>Num_Lock</td>
<td>65407</td>
<td>0xff7f</td>
</tr>
<tr>
<td>KP_Space</td>
<td>65408</td>
<td>0xff80</td>
</tr>
<tr>
<td>KP_Tab</td>
<td>65417</td>
<td>0xff89</td>
</tr>
<tr>
<td>KP_Enter</td>
<td>65421</td>
<td>0xff8d</td>
</tr>
<tr>
<td>KP_F1</td>
<td>65425</td>
<td>0xff91</td>
</tr>
<tr>
<td>KP_F2</td>
<td>65426</td>
<td>0xff92</td>
</tr>
<tr>
<td>Key</td>
<td>Code</td>
<td>ASCII</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>KP_F3</td>
<td>65427</td>
<td>0xff93</td>
</tr>
<tr>
<td>KP_F4</td>
<td>65428</td>
<td>0xff94</td>
</tr>
<tr>
<td>KP_Multiply</td>
<td>65450</td>
<td>0xffaa</td>
</tr>
<tr>
<td>KP_Add</td>
<td>65451</td>
<td>0xffab</td>
</tr>
<tr>
<td>KP_Separator</td>
<td>65452</td>
<td>0xffac</td>
</tr>
<tr>
<td>KP_Subtract</td>
<td>65453</td>
<td>0xffad</td>
</tr>
<tr>
<td>KP_Decimal</td>
<td>65454</td>
<td>0xffae</td>
</tr>
<tr>
<td>KP_Divide</td>
<td>65455</td>
<td>0xffaf</td>
</tr>
<tr>
<td>KP_0</td>
<td>65456</td>
<td>0xffb0</td>
</tr>
<tr>
<td>KP_1</td>
<td>65457</td>
<td>0xffb1</td>
</tr>
<tr>
<td>KP_2</td>
<td>65458</td>
<td>0xffb2</td>
</tr>
<tr>
<td>KP_3</td>
<td>65459</td>
<td>0xffb3</td>
</tr>
<tr>
<td>KP_4</td>
<td>65460</td>
<td>0xffb4</td>
</tr>
<tr>
<td>KP_5</td>
<td>65461</td>
<td>0xffb5</td>
</tr>
<tr>
<td>KP_6</td>
<td>65462</td>
<td>0xffb6</td>
</tr>
<tr>
<td>KP_7</td>
<td>65463</td>
<td>0xffb7</td>
</tr>
<tr>
<td>KP_8</td>
<td>65464</td>
<td>0xffb8</td>
</tr>
<tr>
<td>KP_9</td>
<td>65465</td>
<td>0xffb9</td>
</tr>
<tr>
<td>KP_Equal</td>
<td>65469</td>
<td>0xffbd</td>
</tr>
<tr>
<td>F1</td>
<td>65470</td>
<td>0xffbe</td>
</tr>
<tr>
<td>F2</td>
<td>65471</td>
<td>0xffbf</td>
</tr>
<tr>
<td>F3</td>
<td>65472</td>
<td>0xffc0</td>
</tr>
<tr>
<td>F4</td>
<td>65473</td>
<td>0xffc1</td>
</tr>
<tr>
<td>F5</td>
<td>65474</td>
<td>0xffc2</td>
</tr>
<tr>
<td>F6</td>
<td>65475</td>
<td>0xffc3</td>
</tr>
<tr>
<td>F7</td>
<td>65476</td>
<td>0xffc4</td>
</tr>
<tr>
<td>F8</td>
<td>65477</td>
<td>0xffc5</td>
</tr>
<tr>
<td>F9</td>
<td>65478</td>
<td>0xffc6</td>
</tr>
<tr>
<td>F10</td>
<td>65479</td>
<td>0xffc7</td>
</tr>
<tr>
<td>L1</td>
<td>65480</td>
<td>0xffc8</td>
</tr>
<tr>
<td>L2</td>
<td>65481</td>
<td>0xffc9</td>
</tr>
<tr>
<td>L3</td>
<td>65482</td>
<td>0xffca</td>
</tr>
<tr>
<td>L4</td>
<td>65483</td>
<td>0xffcb</td>
</tr>
<tr>
<td>L5</td>
<td>65484</td>
<td>0xffcc</td>
</tr>
<tr>
<td>L6</td>
<td>65485</td>
<td>0xffcd</td>
</tr>
<tr>
<td>L7</td>
<td>65486</td>
<td>0xffce</td>
</tr>
<tr>
<td>L8</td>
<td>65487</td>
<td>0xffcf</td>
</tr>
<tr>
<td>Key</td>
<td>Code</td>
<td>Value</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>Shift_L</td>
<td>65505</td>
<td>0xffe1</td>
</tr>
<tr>
<td>Shift_R</td>
<td>65506</td>
<td>0xffe2</td>
</tr>
<tr>
<td>Control_L</td>
<td>65507</td>
<td>0xffe3</td>
</tr>
<tr>
<td>Control_R</td>
<td>65508</td>
<td>0xffe4</td>
</tr>
<tr>
<td>Caps_Lock</td>
<td>65509</td>
<td>0xffe5</td>
</tr>
<tr>
<td>Shift_Lock</td>
<td>65510</td>
<td>0xffe6</td>
</tr>
<tr>
<td>Meta_L</td>
<td>65511</td>
<td>0xffe7</td>
</tr>
<tr>
<td>Meta_R</td>
<td>65512</td>
<td>0xffe8</td>
</tr>
<tr>
<td>Alt_L</td>
<td>65513</td>
<td>0xffe9</td>
</tr>
<tr>
<td>Alt_R</td>
<td>65514</td>
<td>0xffea</td>
</tr>
<tr>
<td>Super_L</td>
<td>65515</td>
<td>0xffeb</td>
</tr>
<tr>
<td>Super_R</td>
<td>65516</td>
<td>0xffec</td>
</tr>
<tr>
<td>Hyper_L</td>
<td>65517</td>
<td>0xffed</td>
</tr>
<tr>
<td>Hyper_R</td>
<td>65518</td>
<td>0xffee</td>
</tr>
<tr>
<td>Delete</td>
<td>65535</td>
<td>0xffff</td>
</tr>
</tbody>
</table>

**SEE ALSO**

bind
KEYWORDS

keysym, bind, binding
NAME
radiobutton - Create and manipulate radiobutton widgets

SYNOPSIS

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

WIDGET-SPECIFIC OPTIONS

-command, command, Command
-height, height, Height
-indicatoron, indicatorOn, IndicatorOn
-selectcolor, selectColor, Background
-offrelief, offRelief, OffRelief
-overrelief, overRelief, OverRelief
-selectimage, selectImage, SelectImage
-state, state, State
-tristateimage, tristatimage, TristatImage
-tristatevalue, tristateValue, Value
-value, value, Value
-variable, variable, Variable
-width, width, Width

DESCRIPTION
WIDGET COMMAND

pathName cget option
pathName configure ?option? ?value option value ...
pathName deselect
pathName flash
pathName invoke
pathName select

BINDINGS
SEE ALSO
KEYWORDS

NAME

radiobutton - Create and manipulate radiobutton widgets

SYNOPSIS

radiobutton pathName ?options?

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: **-command**
Database Name: **command**
Database Class: **Command**

Specifies a Tcl command to associate with the button. This command is typically invoked when mouse button 1 is released over the button window. The button's global variable ( -variable option) will be updated before the command is invoked.

Command-Line Name: **-height**
Database Name: **height**
Database Class: **Height**

Specifies a desired height for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to **Tk_GetPixels**); for text it is in lines
of text. If this option is not specified, the button's desired height is computed from the size of the image or bitmap or text being displayed in it.

Command-Line Name: -indicatoron
Database Name: indicatorOn
Database Class: IndicatorOn
  Specifies whether or not the indicator should be drawn. Must be a proper boolean value. If false, the relief option is ignored and the widget's relief is always sunken if the widget is selected and raised otherwise.

Command-Line Name: -selectcolor
Database Name: selectColor
Database Class: Background
  Specifies a background color to use when the button is selected. If indicatorOn is true then the color applies to the indicator. Under Windows, this color is used as the background for the indicator regardless of the select state. If indicatorOn is false, this color is used as the background for the entire widget, in place of background or activeBackground, whenever the widget is selected. If specified as an empty string then no special color is used for displaying when the widget is selected.

Command-Line Name: -offrelief
Database Name: offRelief
Database Class: OffRelief
  Specifies the relief for the checkbutton when the indicator is not drawn and the checkbutton is off. The default value is “raised”. By setting this option to “flat” and setting indicatorOn to false and -overrelief to “raised”, the effect is achieved of having a flat button that raises on mouse-over and which is depressed when activated. This is the behavior typically exhibited by the Align-Left, Align-Right, and Center radiobuttons on the toolbar of a word-processor, for example.

Command-Line Name: -overrelief
Database Name: overRelief
Database Class: **OverRelief**
Specifies an alternative relief for the radiobutton, to be used when the mouse cursor is over the widget. This option can be used to make toolbar buttons, by configuring **-relief flat -overrelief raised**. If the value of this option is the empty string, then no alternative relief is used when the mouse cursor is over the radiobutton. The empty string is the default value.

Command-Line Name: **-selectimage**
Database Name: **selectImage**
Database Class: **SelectImage**
Specifies an image to display (in place of the image option) when the radiobutton is selected. This option is ignored unless the image option has been specified.

Command-Line Name: **-state**
Database Name: **state**
Database Class: **State**
Specifies one of three states for the radiobutton: **normal**, **active**, or **disabled**. In normal state the radiobutton is displayed using the foreground and background options. The active state is typically used when the pointer is over the radiobutton. In active state the radiobutton is displayed using the activeForeground and activeBackground options. Disabled state means that the radiobutton should be insensitive: the default bindings will refuse to activate the widget and will ignore mouse button presses. In this state the disabledForeground and background options determine how the radiobutton is displayed.

Command-Line Name: **-tristateimage**
Database Name: **tristateImage**
Database Class: **TristateImage**
Specifies an image to display (in place of the image option) when the radiobutton is selected. This option is ignored unless the image option has been specified.

Command-Line Name: **-tristatevalue**
Database Name: **tristateValue**
Database Class: **Value**
Specifies the value that causes the radiobutton to display the multi-value selection, also known as the tri-state mode. Defaults to “”.

Command-Line Name: `-value`
Database Name: `value`

Database Class: **Value**
Specifies value to store in the button’s associated variable whenever this button is selected.

Command-Line Name: `-variable`
Database Name: `variable`

Database Class: **Variable**
Specifies name of global variable to set whenever this button is selected. Changes in this variable also cause the button to select or deselect itself. Defaults to the value `selectedButton`.

Command-Line Name: `-width`
Database Name: `width`

Database Class: **Width**
Specifies a desired width for the button. If an image or bitmap is being displayed in the button, the value is in screen units (i.e. any of the forms acceptable to `Tk_GetPixels`); for text it is in characters. If this option is not specified, the button’s desired width is computed from the size of the image or bitmap or text being displayed in it.

**DESCRIPTION**

The **radiobutton** command creates a new window (given by the `pathName` argument) and makes it into a radiobutton widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the radiobutton such as its colors, font, text, and initial relief. The **radiobutton** command returns its `pathName` argument. At the time this command is invoked, there must not exist a window named `pathName`, but `pathName`'s parent must exist.
A radiobutton is a widget that displays a textual string, bitmap or image and a diamond or circle called an *indicator*. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the `wrapLength` option) and one of the characters may optionally be underlined using the `underline` option. A radiobutton has all of the behavior of a simple button: it can display itself in either of three different ways, according to the `state` option; it can be made to appear raised, sunken, or flat; it can be made to flash; and it invokes a Tcl command whenever mouse button 1 is clicked over the check button.

In addition, radiobuttons can be selected. If a radiobutton is selected, the indicator is normally drawn with a selected appearance, and a Tcl variable associated with the radiobutton is set to a particular value (normally 1). Under Unix, the indicator is drawn with a sunken relief and a special color. Under Windows, the indicator is drawn with a round mark inside. If the radiobutton is not selected, then the indicator is drawn with a deselected appearance, and the associated variable is set to a different value (typically 0). The indicator is drawn without a round mark inside. Typically, several radiobuttons share a single variable and the value of the variable indicates which radiobutton is to be selected. When a radiobutton is selected it sets the value of the variable to indicate that fact; each radiobutton also monitors the value of the variable and automatically selects and deselects itself when the variable's value changes. If the variable's value matches the `tristateValue`, then the radiobutton is drawn using the tri-state mode. This mode is used to indicate mixed or multiple values. (This is used when the radiobutton represents the state of multiple items.) By default the variable `selectedButton` is used; its contents give the name of the button that is selected, or the empty string if no button associated with that variable is selected. The name of the variable for a radiobutton, plus the variable to be stored into it, may be modified with options on the command line or in the option database. Configuration options may also be used to modify the way the indicator is displayed (or whether it is displayed at all). By default a radiobutton is configured to select itself on button clicks.
The **radiobutton** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```plaintext
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for radiobutton widgets:

**pathName cget option**

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **radiobutton** command.

**pathName configure ?option? ?value option value ...??**

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see [Tk ConfigureInfo](#) for information on the format of this list). If *option* is specified with no *value*, the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option-value* pairs are specified, the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **radiobutton** command.

**pathName deselect**

Deselects the radiobutton and sets the associated variable to an empty string. If this radiobutton was not currently selected, the command has no effect.

**pathName flash**

Flashes the radiobutton. This is accomplished by redisplaying the
radiobutton several times, alternating between active and normal colors. At the end of the flash the radiobutton is left in the same normal/active state as when the command was invoked. This command is ignored if the radiobutton's state is disabled.

\textit{pathName invoke}

Does just what would have happened if the user invoked the radiobutton with the mouse: selects the button and invokes its associated Tcl command, if there is one. The return value is the return value from the Tcl command, or an empty string if there is no command associated with the radiobutton. This command is ignored if the radiobutton's state is disabled.

\textit{pathName select}

Selects the radiobutton and sets the associated variable to the value corresponding to this widget.

\begin{tabular}{|l|}
\hline
\textbf{BINDINGS} \\
\hline
\end{tabular}

Tk automatically creates class bindings for radiobuttons that give them the following default behavior:

[1]

On Unix systems, a radiobutton activates whenever the mouse passes over it and deactivates whenever the mouse leaves the radiobutton. On Mac and Windows systems, when mouse button 1 is pressed over a radiobutton, the button activates whenever the mouse pointer is inside the button, and deactivates whenever the mouse pointer leaves the button.

[2]

When mouse button 1 is pressed over a radiobutton it is invoked (it becomes selected and the command associated with the button is invoked, if there is one).

[3]

When a radiobutton has the input focus, the space key causes the radiobutton to be invoked.
If the radiobutton's state is **disabled** then none of the above actions occur: the radiobutton is completely non-responsive.

The behavior of radiobuttons can be changed by defining new bindings for individual widgets or by redefining the class bindings.

**SEE ALSO**

[checkbutton](#), [labelframe](#), [listbox](#), [options](#), [scale](#), [ttk::radiobutton](#)

**KEYWORDS**

radiobutton, widget

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
   ttk::button - Widget that issues a command when pressed

SYNOPSIS
DESCRIPTION
STANDARD OPTIONS
   -class
   -compound, compound, Compound
   -cursor, cursor, Cursor
   -image, image, Image
   -state
   -style
   -takefocus, takeFocus, TakeFocus
   -text, text, Text
   -textvariable, textVariable, Variable
   -underline, underline, Underline
   -width

WIDGET-SPECIFIC OPTIONS
   -command, command, Command
   -default, default, Default
   -width, width, Width

WIDGET COMMAND
   pathName invoke

COMPATIBILITY OPTIONS
   -state, state, State

SEE ALSO

KEYWORDS
SYNOPSIS

ttk::button pathName ?options?

DESCRIPTION

A ttk::button widget displays a textual label and/or image, and evaluates a command when pressed.

STANDARD OPTIONS

-class
-compound, compound, Compound
-cursor, cursor, Cursor
-image, image, Image
-state
-style
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-width

WIDGET-SPECIFIC OPTIONS

Command-Line Name: -command
Database Name: command
Database Class: Command
   A script to evaluate when the widget is invoked.

Command-Line Name: -default
Database Name: default
Database Class: Default
   May be set to one of normal, active, or disabled. In a dialog box, one button may be designated the “default” button (meaning, roughly, “the one that gets invoked when the user presses <Enter>”). active indicates that this is currently the default button; normal means that it may become the default button, and disabled
means that it is not defaultable. The default is normal.

Depending on the theme, the default button may be displayed with an extra highlight ring, or with a different border color.

Command-Line Name: -width
Database Name: width
Database Class: Width
If greater than zero, specifies how much space, in character widths, to allocate for the text label. If less than zero, specifies a minimum width. If zero or unspecified, the natural width of the text label is used. Note that some themes may specify a non-zero -width in the style.

WIDGET COMMAND

In addition to the standard cget, configure, identify, instate, and state commands, buttons support the following additional widget commands:

pathName invoke
Invokes the command associated with the button.

COMPATIBILITY OPTIONS

Command-Line Name: -state
Database Name: state
Database Class: State
May be set to normal or disabled to control the disabled state bit. This is a “write-only” option: setting it changes the widget state, but the state widget command does not affect the state option.

SEE ALSO

ttk::widget, button

KEYWORDS

widget, button, default, command
NAME

ttk::separator - Separator bar

SYNOPSIS

ttk::separator pathName ?options?

DESCRIPTION

A ttk::separator widget displays a horizontal or vertical separator bar.

STANDARD OPTIONS

-class
-cursor, cursor, Cursor
-state
-style
-takefocus, takeFocus, TakeFocus
WIDGET-SPECIFIC OPTIONS

Command-Line Name: -orient
Database Name: orient
Database Class: Orient

One of horizontal or vertical. Specifies the orientation of the separator.

WIDGET COMMAND

Separator widgets support the standard cget, configure, identify, instate, and state methods. No other widget methods are used.

SEE ALSO

ttk::widget

KEYWORDS

widget, separator

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2004 Joe English
NAME
checkbutton - Create and manipulate checkbutton widgets

SYNOPSIS

STANDARD OPTIONS
- -activebackground, activeBackground, Foreground
- -activeforeground, activeForeground, Background
- -anchor, anchor, Anchor
- -background or -bg, background, Background
- -bitmap, bitmap, Bitmap
- -borderwidth or -bd, borderWidth, BorderWidth
- -compound, compound, Compound
- -cursor, cursor, Cursor
- -disabledforeground, disabledForeground, DisabledForeground
- -font, font, Font
- -foreground or -fg, foreground, Foreground
- -highlightbackground, highlightBackground, HighlightBackground
- -highlightcolor, highlightColor, HighlightColor
- -highlightthickness, highlightThickness, HighlightThickness
- -image, image, Image
- -justify, justify, Justify
- -padx, padX, Pad
- -pady, padY, Pad
- -relief, relief, Relief
- -takefocus, takeFocus, TakeFocus
- -text, text, Text
- -textvariable, textVariable, Variable
- -underline, underline, Underline
- -wraplength, wrapLength, WrapLength

WIDGET-SPECIFIC OPTIONS
- -command, command, Command
-height, height, Height
-indicatoron, indicatorOn, IndicatorOn
-offrelief, offRelief, OffRelief
-offvalue, offValue, Value
-onvalue, onValue, Value
-overrelief, overRelief, OverRelief
-selectcolor, selectColor, Background
-selectimage, selectImage, SelectImage
-state, state, State
-tristateimage, tristateImage, TristateImage
-tristatevalue, tristateValue, Value
-variable, variable, Variable
-width, width, Width

DESCRIPTION

WIDGET COMMAND

`pathName cget option`
`pathName configure ?option? ?value option value ...?`
`pathName deselect`
`pathName flash`
`pathName invoke`
`pathName select`
`pathName toggle`

BINDINGS
EXAMPLE
SEE ALSO
KEYWORDS

<table>
<thead>
<tr>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>checkbutton - Create and manipulate checkbutton widgets</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYNOPSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>checkbutton</strong> <em>pathName</em> <em>?options</em>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STANDARD OPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>-activebackground, activeBackground, Foreground</strong></td>
</tr>
</tbody>
</table>
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

<table>
<thead>
<tr>
<th>WIDGET-SPECIFIC OPTIONS</th>
</tr>
</thead>
</table>

Command-Line Name: **-command**
Database Name: **command**
Database Class: **Command**

Specifies a Tcl command to associate with the button. This command is typically invoked when mouse button 1 is released over the button window. The button's global variable (**-variable** option) will be updated before the command is invoked.

Command-Line Name: **-height**
Database Name: **height**
Database Class: **Height**
Specifies a desired height for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to Tk::GetPixels); for text it is in lines of text. If this option is not specified, the button’s desired height is computed from the size of the image or bitmap or text being displayed in it.

Command-Line Name: -indicatoron
Database Name: indicatorOn
Database Class: IndicatorOn
   Specifies whether or not the indicator should be drawn. Must be a proper boolean value. If false, the relief option is ignored and the widget’s relief is always sunken if the widget is selected and raised otherwise.

Command-Line Name: -offrelief
Database Name: offRelief
Database Class: OffRelief
   Specifies the relief for the checkbutton when the indicator is not drawn and the checkbutton is off. The default value is “raised”. By setting this option to “flat” and setting -indicatoron to false and -overrelief to “raised”, the effect is achieved of having a flat button that raises on mouse-over and which is depressed when activated. This is the behavior typically exhibited by the Bold, Italic, and Underline checkbuttons on the toolbar of a word-processor, for example.

Command-Line Name: -offvalue
Database Name: offValue
Database Class: Value
   Specifies value to store in the button's associated variable whenever this button is deselected. Defaults to “0”.

Command-Line Name: -onvalue
Database Name: onValue
Database Class: Value
   Specifies value to store in the button's associated variable whenever this button is selected. Defaults to “1”.
Command-Line Name: -overrelief
Database Name: overRelief
Database Class: OverRelief
Specifies an alternative relief for the checkbutton, to be used when the mouse cursor is over the widget. This option can be used to make toolbar buttons, by configuring -relief flat -overrelief raised. If the value of this option is the empty string, then no alternative relief is used when the mouse cursor is over the checkbutton. The empty string is the default value.

Command-Line Name: -selectcolor
Database Name: selectColor
Database Class: Background
Specifies a background color to use when the button is selected. If indicatorOn is true then the color is used as the background for the indicator regardless of the select state. If indicatorOn is false, this color is used as the background for the entire widget, in place of background or activeBackground, whenever the widget is selected. If specified as an empty string then no special color is used for displaying when the widget is selected.

Command-Line Name: -selectimage
Database Name: selectImage
Database Class: SelectImage
Specifies an image to display (in place of the image option) when the checkbutton is selected. This option is ignored unless the image option has been specified.

Command-Line Name: -state
Database Name: state
Database Class: State
Specifies one of three states for the checkbutton: normal, active, or disabled. In normal state the checkbutton is displayed using the foreground and background options. The active state is typically used when the pointer is over the checkbutton. In active state the checkbutton is displayed using the activeForeground and activeBackground options. Disabled state means that the checkbutton should be insensitive: the default bindings will refuse
to activate the widget and will ignore mouse button presses. In this state the disabledForeground and background options determine how the checkbutton is displayed.

Command-Line Name: **-tristateimage**
Database Name: **tristatImage**
Database Class: **TristateImage**
    Specifies an image to display (in place of the image option) when the checkbutton is in tri-state mode. This option is ignored unless the image option has been specified.

Command-Line Name: **-tristatevalue**
Database Name: **tristateValue**
Database Class: **Value**
    Specifies the value that causes the checkbutton to display the multi-value selection, also known as the tri-state mode. Defaults to "".

Command-Line Name: **-variable**
Database Name: **variable**
Database Class: **Variable**
    Specifies name of global variable to set to indicate whether or not this button is selected. Defaults to the name of the button within its parent (i.e. the last element of the button window's path name).

Command-Line Name: **-width**
Database Name: **width**
Database Class: **Width**
    Specifies a desired width for the button. If an image or bitmap is being displayed in the button then the value is in screen units (i.e. any of the forms acceptable to **Tk_GetPixels**); for text it is in characters. If this option is not specified, the button's desired width is computed from the size of the image or bitmap or text being displayed in it.

---

**DESCRIPTION**

The **checkbutton** command creates a new window (given by the
pathName argument) and makes it into a checkbutton widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the checkbutton such as its colors, font, text, and initial relief. The checkbutton command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

A checkbutton is a widget that displays a textual string, bitmap or image and a square called an indicator. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the wrapLength option) and one of the characters may optionally be underlined using the underline option. A checkbutton has all of the behavior of a simple button, including the following: it can display itself in either of three different ways, according to the state option; it can be made to appear raised, sunken, or flat; it can be made to flash; and it invokes a Tcl command whenever mouse button 1 is clicked over the checkbutton.

In addition, checkbuttons can be selected. If a checkbutton is selected then the indicator is normally drawn with a selected appearance, and a Tcl variable associated with the checkbutton is set to a particular value (normally 1). The indicator is drawn with a check mark inside. If the checkbutton is not selected, then the indicator is drawn with a deselected appearance, and the associated variable is set to a different value (typically 0). The indicator is drawn without a check mark inside. In the special case where the variable (if specified) has a value that matches the tristatevalue, the indicator is drawn with a tri-state appearance and is in the tri-state mode indicating mixed or multiple values. (This is used when the check box represents the state of multiple items.) The indicator is drawn in a platform dependent manner. Under Unix and Windows, the background interior of the box is “grayed”. Under Mac, the indicator is drawn with a dash mark inside. By default, the name of the variable associated with a checkbutton is the same as the name used to create the checkbutton. The variable name, and the “on”, “off” and “tristate” values stored in it, may be modified with options on the command line or in the option database. Configuration
options may also be used to modify the way the indicator is displayed (or whether it is displayed at all). By default a checkbutton is configured to select and deselect itself on alternate button clicks. In addition, each checkbutton monitors its associated variable and automatically selects and deselects itself when the variables value changes to and from the button's “on”, “off” and “tristate” values.

**WIDGET COMMAND**

The **checkbutton** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for checkbutton widgets:

**pathName cget option**

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **checkbutton** command.

**pathName configure ?option? ?value option value ...?**

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option-value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **checkbutton** command.
pathName deselect
Deselects the checkbutton and sets the associated variable to its “off” value.

pathName flash
Flashes the checkbutton. This is accomplished by redisplaying the checkbutton several times, alternating between active and normal colors. At the end of the flash the checkbutton is left in the same normal/active state as when the command was invoked. This command is ignored if the checkbutton's state is disabled.

pathName invoke
Does just what would have happened if the user invoked the checkbutton with the mouse: toggle the selection state of the button and invoke the Tcl command associated with the checkbutton, if there is one. The return value is the return value from the Tcl command, or an empty string if there is no command associated with the checkbutton. This command is ignored if the checkbutton's state is disabled.

pathName select
Selects the checkbutton and sets the associated variable to its “on” value.

pathName toggle
Toggles the selection state of the button, redisplaying it and modifying its associated variable to reflect the new state.

BINDINGS

Tk automatically creates class bindings for checkbuttons that give them the following default behavior:

[1]
On Unix systems, a checkbutton activates whenever the mouse passes over it and deactivates whenever the mouse leaves the checkbutton. On Mac and Windows systems, when mouse button 1 is pressed over a checkbutton, the button activates whenever the
mouse pointer is inside the button, and deactivates whenever the mouse pointer leaves the button.

[2] When mouse button 1 is pressed over a checkbox, it is invoked (its selection state toggles and the command associated with the button is invoked, if there is one).

[3] When a checkbox has the input focus, the space key causes the checkbox to be invoked. Under Windows, there are additional key bindings; plus (+) and equal (=) select the button, and minus (-) deselects the button.

If the checkbox's state is disabled then none of the above actions occur: the checkbox is completely non-responsive.

The behavior of checkboxes can be changed by defining new bindings for individual widgets or by redefining the class bindings.

**EXAMPLE**

This example shows a group of uncoupled checkboxes.

```python
labelframe .lbl -text "Steps:"
    checkbutton .c1 -text Lights -variable lights
    checkbutton .c2 -text Cameras -variable cameras
    checkbutton .c3 -text Action! -variable action
pack .c1 .c2 .c3 -in .lbl
pack .lbl
```

**SEE ALSO**

`button`, `options`, `radiobutton`, `ttk::checkbutton`

**KEYWORDS**
checkbutton, widget
NAME
label - Create and manipulate label widgets

SYNOPSIS

STANDARD OPTIONS
- activebackground, activeBackground, Foreground
- activeforeground, activeForeground, Background
- anchor, anchor, Anchor
- background or -bg, background, Background
- bitmap, bitmap, Bitmap
- borderwidth or -bd, borderWidth, BorderWidth
- compound, compound, Compound
- cursor, cursor, Cursor
- disabledforeground, disabledForeground, DisabledForeground
- font, font, Font
- foreground or -fg, foreground, Foreground
- highlightbackground, highlightBackground, HighlightBackground
- highlightcolor, highlightColor, HighlightColor
- highlightthickness, highlightThickness, HighlightThickness
- image, image, Image
- justify, justify, Justify
- padx, padX, Pad
- pady, padY, Pad
- relief, relief, Relief
- takefocus, takeFocus, TakeFocus
- text, text, Text
- textvariable, textVariable, Variable
- underline, underline, Underline
- wraplength, wrapLength, WrapLength

WIDGET-SPECIFIC OPTIONS
- height, height, Height
-state, state, State
-width, width, Width

DESCRIPTION
WIDGET COMMAND

\texttt{pathName cget option}
\texttt{pathName configure ?option? ?value option value ...?}

BINDINGS
EXAMPLE
SEE ALSO
KEYWORDS

\begin{tabular}{|l|}
\hline
\textbf{NAME} \\
\hline
label - Create and manipulate label widgets \\
\hline
\textbf{SYNOPSIS} \\
\hline
\textbf{label} \textit{pathName} \textit{?options?} \\
\hline
\textbf{STANDARD OPTIONS} \\
\hline
-\texttt{activebackground}, \texttt{activeBackground}, Foreground
-\texttt{activeforeground}, \texttt{activeForeground}, Background
-\texttt{anchor}, \texttt{anchor}, Anchor
-\texttt{background or -bg}, \texttt{background}, Background
-\texttt{bitmap}, \texttt{bitmap}, Bitmap
-\texttt{borderwidth or -bd}, \texttt{borderWidth}, BorderWidth
-\texttt{compound}, \texttt{compound}, Compound
-\texttt{cursor}, \texttt{cursor}, Cursor
-\texttt{disabledforeground}, \texttt{disabledForeground}, DisabledForeground
-\texttt{font}, \texttt{font}, Font
-\texttt{foreground or -fg}, \texttt{foreground}, Foreground
-\texttt{highlightbackground}, \texttt{highlightBackground}, HighlightBackground
-\texttt{highlightcolor}, \texttt{highlightColor}, HighlightColor
-\texttt{highlightthickness}, \texttt{highlightThickness}, HighlightThickness
-\texttt{image}, \texttt{image}, Image
-\texttt{justify}, \texttt{justify}, Justify
-\texttt{padx}, \texttt{padX}, Pad
\hline
\end{tabular}
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

WIDGET-SPECIFIC OPTIONS

Command-Line Name: **-height**
Database Name: **height**
Database Class: **Height**
   Specifies a desired height for the label. If an image or bitmap is being displayed in the label then the value is in screen units (i.e. any of the forms acceptable to **Tk_GetPixels**); for text it is in lines of text. If this option is not specified, the label's desired height is computed from the size of the image or bitmap or text being displayed in it.

Command-Line Name: **-state**
Database Name: **state**
Database Class: **State**
   Specifies one of three states for the label: **normal**, **active**, or **disabled**. In normal state the button is displayed using the **foreground** and **background** options. In active state the label is displayed using the **activeForeground** and **activeBackground** options. In the disabled state the **disabledForeground** and **background** options determine how the button is displayed.

Command-Line Name: **-width**
Database Name: **width**
Database Class: **Width**
   Specifies a desired width for the label. If an image or bitmap is being displayed in the label then the value is in screen units (i.e. any of the forms acceptable to **Tk_GetPixels**); for text it is in characters. If this option is not specified, the label's desired width is computed from the size of the image or bitmap or text being
displayed in it.

**DESCRIPTION**

The **label** command creates a new window (given by the *pathName* argument) and makes it into a label widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the label such as its colors, font, text, and initial relief. The **label** command returns its *pathName* argument. At the time this command is invoked, there must not exist a window named *pathName*, but *pathName*'s parent must exist.

A label is a widget that displays a textual string, bitmap or image. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the **wrapLength** option) and one of the characters may optionally be underlined using the **underline** option. The label can be manipulated in a few simple ways, such as changing its relief or text, using the commands described below.

**WIDGET COMMAND**

The **label** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for label widgets:

*pathName cget option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **label** command.

*pathName configure ?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option-value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the label command.

BINDINGS

When a new label is created, it has no default event bindings: labels are not intended to be interactive.

EXAMPLE

```python
# Make the widgets
label .t -text "This widget is at the top" -bg red
label .b -text "This widget is at the bottom" -bg green
label .l -text "Left\nHand\nSide"
label .r -text "Right\nHand\nSide"
text .mid
.mid insert end "This layout is like Java's BorderLayout"
# Lay them out
pack .t -side top -fill x
pack .b -side bottom -fill x
pack .l -side left -fill y
pack .r -side right -fill y
pack .mid -expand 1 -fill both
```

SEE ALSO
labelframe, button, ttk::label

KEYWORDS

cos, widget

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
**NAME**

raise - Change a window's position in the stacking order

**SYNOPSIS**

`raise window ?aboveThis?`

**DESCRIPTION**

If the `aboveThis` argument is omitted then the command raises `window` so that it is above all of its siblings in the stacking order (it will not be obscured by any siblings and will obscure any siblings that overlap it). If `aboveThis` is specified then it must be the path name of a window that is either a sibling of `window` or the descendant of a sibling of `window`. In this case the `raise` command will insert `window` into the stacking order just above `aboveThis` (or the ancestor of `aboveThis` that is a sibling of `window`); this could end up either raising or lowering `window`.

**EXAMPLE**

Make a button appear to be in a sibling frame that was created after it. This is is often necessary when building GUIs in the style where you create your activity widgets first before laying them out on the display:

```tcl
button .b -text "Hi there!"
pack [frame .f -background blue]
pack [label .f.l1 -text "This is above"]
pack .b -in .f
pack [label .f.l2 -text "This is below"]
raise .b
```
SEE ALSO

lower

KEYWORDS

obscure, raise, stacking order
NAME

tk_getOpenFile, tk_getSaveFile - pop up a dialog box for the user to select a file to open or save.

SYNOPSIS

DESCRIPTION

 OSError

SPECIFYING FILE PATTERNS

SPECIFYING EXTENSIONS

EXAMPLE

SEE ALSO

KEYWORDS

NAME

tk_getOpenFile, tk_getSaveFile - pop up a dialog box for the user to select a file to open or save.

SYNOPSIS

tk_getOpenFile ?option value ...?
tk_getSaveFile ?option value ...?
The procedures `tk_getOpenFile` and `tk_getSaveFile` pop up a dialog box for the user to select a file to open or save. The `tk_getOpenFile` command is usually associated with the **Open** command in the **File** menu. Its purpose is for the user to select an existing file only. If the user enters a non-existent file, the dialog box gives the user an error prompt and requires the user to give an alternative selection. If an application allows the user to create new files, it should do so by providing a separate **New** menu command.

The `tk_getSaveFile` command is usually associated with the **Save as** command in the **File** menu. If the user enters a file that already exists, the dialog box prompts the user for confirmation whether the existing file should be overwritten or not.

The following *option-value* pairs are possible as command line arguments to these two commands:

- **-defaultextension** *extension*
  Specifies a string that will be appended to the filename if the user enters a filename without an extension. The default value is the empty string, which means no extension will be appended to the filename in any case. This option is ignored on Mac OS X, which does not require extensions to filenames, and the UNIX implementation guesses reasonable values for this from the **-filetypes** option when this is not supplied.

- **-filetypes** *filePatternList*
  If a **File types** listbox exists in the file dialog on the particular platform, this option gives the **filetypes** in this listbox. When the user choose a filetype in the listbox, only the files of that type are listed. If this option is unspecified, or if it is set to the empty list, or if the **File types** listbox is not supported by the particular platform then all files are listed regardless of their types. See the section **SPECIFYING FILE PATTERNS** below for a discussion on the contents of *filePatternList*. 
-**initialdir** *directory*
  Specifies that the files in *directory* should be displayed when the dialog pops up. If this parameter is not specified, then the files in the current working directory are displayed. If the parameter specifies a relative path, the return value will convert the relative path to an absolute path.

-**initialfile** *filename*
  Specifies a filename to be displayed in the dialog when it pops up.

-**message** *string*
  Specifies a message to include in the client area of the dialog. This is only available on Mac OS X.

-**multiple** *boolean*
  Allows the user to choose multiple files from the Open dialog.

-**parent** *window*
  Makes *window* the logical parent of the file dialog. The file dialog is displayed on top of its parent window. On Mac OS X, this turns the file dialog into a sheet attached to the parent window.

-**title** *titleString*
  Specifies a string to display as the title of the dialog box. If this option is not specified, then a default title is displayed.

-**typevariable** *variableName*
  The global variable *variableName* is used to preselect which filter is used from *filterList* when the dialog box is opened and is updated when the dialog box is closed, to the last selected filter. The variable is read once at the beginning to select the appropriate filter. If the variable does not exist, or its value does not match any filter typename, or is empty ({}), the dialog box will revert to the default behavior of selecting the first filter in the list. If the dialog is canceled, the variable is not modified.

If the user selects a file, both **tk_getOpenFile** and **tk_getSaveFile** return the full pathname of this file. If the user cancels the operation,
both commands return the empty string.

**SPECIFYING FILE PATTERNS**

The `filePatternList` value given by the `-filetypes` option is a list of file patterns. Each file pattern is a list of the form

```
typeName {extension ?extension ...?} ?{macType ?macT...
```

`typeName` is the name of the file type described by this file pattern and is the text string that appears in the **File types** listbox. `extension` is a file extension for this file pattern. `macType` is a four-character Macintosh file type. The list of `macTypes` is optional and may be omitted for applications that do not need to execute on the Macintosh platform.

Several file patterns may have the same `typeName`, in which case they refer to the same file type and share the same entry in the listbox. When the user selects an entry in the listbox, all the files that match at least one of the file patterns corresponding to that entry are listed. Usually, each file pattern corresponds to a distinct type of file. The use of more than one file pattern for one type of file is only necessary on the Macintosh platform.

On the Macintosh platform, a file matches a file pattern if its name matches at least one of the `extension`(s) AND it belongs to at least one of the `macType`(s) of the file pattern. For example, the **C Source Files** file pattern in the sample code matches with files that have a `.c` extension AND belong to the `macType` **TEXT**. To use the OR rule instead, you can use two file patterns, one with the `extensions` only and the other with the `macType` only. The **GIF Files** file type in the sample code matches files that **either** have a `.gif` extension OR belong to the `macType` **GIFF**.

On the Unix and Windows platforms, a file matches a file pattern if its name matches at least one of the `extension`(s) of the file pattern. The `macTypes` are ignored.
SPECIFYING EXTENSIONS

On the Unix and Macintosh platforms, extensions are matched using glob-style pattern matching. On the Windows platform, extensions are matched by the underlying operating system. The types of possible extensions are:

(1) the special extension “*” matches any file;

(2) the special extension “” matches any files that do not have an extension (i.e., the filename contains no full stop character);

(3) any character string that does not contain any wild card characters (* and ?).

Due to the different pattern matching rules on the various platforms, to ensure portability, wild card characters are not allowed in the extensions, except as in the special extension “*”. Extensions without a full stop character (e.g. “~”) are allowed but may not work on all platforms.

EXAMPLE

```bash
set types {
    {{Text Files} {.txt} }
    {{TCL Scripts} {.tcl} }
    {{C Source Files} {.c} TEXT}
    {{GIF Files} {.gif} }
    {{GIF Files} {} GIFF}
    {{All Files} *}
}
set filename [tk_getOpenFile -filetypes $types]
if {$filename != ""} {
    # Code goes here
```
# Open the file ...
}

**SEE ALSO**

tk_chooseDirectory

**KEYWORDS**

file selection dialog

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
NAME
ttk::checkbutton - On/off widget

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS
   -class
   -compound, compound, Compound
   -cursor, cursor, Cursor
   -image, image, Image
   -state
   -style
   -takefocus, takeFocus, TakeFocus
   -text, text, Text
   -textvariable, textVariable, Variable
   -underline, underline, Underline
   -width

WIDGET-SPECIFIC OPTIONS
   -command, command, Command
   -offvalue, offValue, OffValue
   -onvalue, onValue, OnValue
   -variable, variable, Variable

WIDGET COMMAND
   pathname invoke

WIDGET STATES

SEE ALSO

KEYWORDS

NAME
ttk::checkbutton - On/off widget
SYNOPSIS

**ttk::checkbutton** *pathName* ?*options*?

DESCRIPTION

A **ttk::checkbutton** widget is used to show or change a setting. It has two states, selected and deselected. The state of the checkbutton may be linked to a Tcl variable.

STANDARD OPTIONS

- **-class**
- **-compound, compound, Compound**
- **-cursor, cursor, Cursor**
- **-image, image, Image**
- **-state**
- **-style**
- **-takefocus, takeFocus, TakeFocus**
- **-text, text, Text**
- **-textvariable, textVariable, Variable**
- **-underline, underline, Underline**
- **-width**

WIDGET-SPECIFIC OPTIONS

Command-Line Name: **-command**
Database Name: **command**
Database Class: **Command**

A Tcl script to execute whenever the widget is invoked.

Command-Line Name: **-offvalue**
Database Name: **offValue**
Database Class: **OffValue**

The value to store in the associated **-variable** when the widget is deselected. Defaults to 0.

Command-Line Name: **-onvalue**
Database Name: **onValue**
Database Class: **OnValue**

The value to store in the associated -variable when the widget is selected. Defaults to 1.

Command-Line Name: **-variable**
Database Name: **variable**
Database Class: **Variable**

The name of a global variable whose value is linked to the widget. Defaults to the widget pathname if not specified.

**WIDGET COMMAND**

In addition to the standard **cget, configure, identify, instate, and state** commands, checkbuttons support the following additional widget commands:

**pathname invoke**

Toggles between the selected and deselected states and evaluates the associated -command. If the widget is currently selected, sets the -variable to the -offvalue and deselects the widget; otherwise, sets the -variable to the -onvalue Returns the result of the -command.

**WIDGET STATES**

The widget does not respond to user input if the **disabled** state is set. The widget sets the **selected** state whenever the linked -variable is set to the widget’s -onvalue, and clears it otherwise. The widget sets the **alternate** state whenever the linked -variable is unset. (The alternate state may be used to indicate a “tri-state” or “indeterminate” selection.)

**SEE ALSO**

**ttk::widget, ttk::radiobutton, checkbutton**

**KEYWORDS**
widget, button, toggle, check, option
NAME

ttk::sizegrip - Bottom-right corner resize widget

SYNOPSIS

**ttk::sizegrip** *pathName* ?*options*?

DESCRIPTION

A **ttk::sizegrip** widget (also known as a *grow box*) allows the user to resize the containing toplevel window by pressing and dragging the grip.

STANDARD OPTIONS

- **-class**
- **-cursor**, **cursor**, **Cursor**
- **-state**
- **-style**
- **-takefocus**, **takeFocus**, **TakeFocus**
-class
-cursor, cursor, Cursor
-state
-style
-takefocus, takeFocus, TakeFocus

## WIDGET COMMAND

Sizegrip widgets support the standard `cget`, `configure`, `identify`, `instate`, and `state` methods. No other widget methods are used.

## PLATFORM-SPECIFIC NOTES

On Mac OSX, toplevel windows automatically include a built-in size grip by default. Adding a `ttk::sizegrip` there is harmless, since the built-in grip will just mask the widget.

## EXAMPLES

Using pack:

```python
pack [ttk::frame $top.statusbar] -side bottom -fill
pack [ttk::sizegrip $top.statusbar.grip] -side right
```

Using grid:

```python
grid [ttk::sizegrip $top.statusbar.grip] \n    -row $lastRow -column $lastColumn -sticky se
# ... optional: add vertical scrollbar in $lastColumn
# ... optional: add horizontal scrollbar in $lastRow
```

## BUGS

If the containing toplevel's position was specified relative to the right or
bottom of the screen (e.g., “\texttt{wm geometry ... wxh-x-y}” instead of “\texttt{wm geometry ... wxh+x+y}”), the sizegrip widget will not resize the window.

\texttt{ttk::sizegrip} widgets only support “southeast” resizing.

\textbf{SEE ALSO}

\texttt{ttk::widget}

\textbf{KEYWORDS}

\texttt{widget, sizegrip, grow box}
clipboard - Manipulate Tk clipboard

DESCRIPTION

This command provides a Tcl interface to the Tk clipboard, which stores data for later retrieval using the selection mechanism (via the -selection CLIPBOARD option). In order to copy data into the clipboard, clipboard clear must be called, followed by a sequence of one or more calls to clipboard append. To ensure that the clipboard is updated atomically, all appends should be completed before returning to the event loop.

The first argument to clipboard determines the format of the rest of the arguments and the behavior of the command. The following forms are currently supported:

clipboard clear ?-displayof window?
    Claims ownership of the clipboard on window's display and removes any previous contents. Window defaults to “.”. Returns an empty string.

clipboard append ?-displayof window? ?-format format? ?-type type? ?--? data
    Appends data to the clipboard on window's display in the form given by type with the representation given by format and claims ownership of the clipboard on window's display.
Type specifies the form in which the selection is to be returned (the desired “target” for conversion, in ICCCM terminology), and should be an atom name such as STRING or FILE_NAME; see the Inter-Client Communication Conventions Manual for complete details. Type defaults to STRING.

The format argument specifies the representation that should be used to transmit the selection to the requester (the second column of Table 2 of the ICCCM), and defaults to STRING. If format is STRING, the selection is transmitted as 8-bit ASCII characters. If format is ATOM, then the data is divided into fields separated by white space; each field is converted to its atom value, and the 32-bit atom value is transmitted instead of the atom name. For any other format, data is divided into fields separated by white space and each field is converted to a 32-bit integer; an array of integers is transmitted to the selection requester. Note that strings passed to clipboard append are concatenated before conversion, so the caller must take care to ensure appropriate spacing across string boundaries. All items appended to the clipboard with the same type must have the same format.

The format argument is needed only for compatibility with clipboard requesters that do not use Tk. If the Tk toolkit is being used to retrieve the CLIPBOARD selection then the value is converted back to a string at the requesting end, so format is irrelevant.

A -- argument may be specified to mark the end of options: the next argument will always be used as data. This feature may be convenient if, for example, data starts with a -.

**EXAMPLES**

```bash
clipboard get -displayof window? -type type?
```

Retrieve data from the clipboard on window's display. Window defaults to “.”. Type specifies the form in which the data is to be returned and should be an atom name such as STRING or FILE_NAME. Type defaults to STRING. This command is equivalent to “selection get -selection CLIPBOARD”.
Get the current contents of the clipboard.

```tcl
if {{[catch {clipboard get} contents]}} {
    # There were no clipboard contents at all
}
```

Set the clipboard to contain a fixed string.

```tcl
clipboard clear
clipboard append "some fixed string"
```

You can put custom data into the clipboard by using a custom `-type` option. This is not necessarily portable, but can be very useful. The method of passing Tcl scripts this way is effective, but should be mixed with safe interpreters in production code.

```tcl
# This is a very simple canvas serializer;
# it produces a script that recreates the item(s) when
proc getItemConfig {canvas tag} {
    set script {}
    foreach item [$canvas find withtag $tag] {
        append script {$canvas create } [$canvas type
        append script { } [$canvas coords $item] { } ]
        foreach config [$canvas itemconf $item] {
            lassign $config name - - - value
            append script [list $name $value] { }
        }
        append script \n    }
    return [string trim $script]
}
```

# Set up a binding on a canvas to cut and paste an i
set c [canvas .c]
pack $c
$c create text 150 30 -text "cut and paste me"
bind $c <<Cut>> {
    clipboard clear
    clipboard append -type TkCanvasItem \[getItemConfig %W current]\n    # Delete because this is cut, not copy. %W delete current
}
bind $c <<Paste>> {
    catch {
        set canvas %W
eval [clipboard get -type TkCanvasItem]
    }
}

SEE ALSO

interp, selection

KEYWORDS

clear, format, clipboard, append, selection, type

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
labelframe - Create and manipulate labelframe widgets

SYNOPSIS
STANDARD OPTIONS
-`borderwidth` or `-bd`, `borderWidth`, `BorderWidth`
-`cursor`, `cursor`, `Cursor`
-`font`, `font`, `Font`
-`foreground` or `-fg`, `foreground`, `Foreground`
-`highlightbackground`, `highlightBackground`
-`highlightcolor`, `highlightColor`, `HighlightColor`
-`highlightthickness`, `highlightThickness`, `HighlightThickness`
-`padX`, `padX`, `Pad`
-`padY`, `padY`, `Pad`
-`relief`, `relief`, `Relief`
-`takefocus`, `takeFocus`, `TakeFocus`
-`text`, `text`, `Text`

WIDGET-SPECIFIC OPTIONS
-`background`, `background`, `Background`
-`class`, `class`, `Class`
-`colormap`, `colormap`, `Colormap`
-`container`, `container`, `Container`
-`height`, `height`, `Height`
-`labelanchor`, `labelAnchor`, `LabelAnchor`
-`labelwidget`, `labelWidget`, `LabelWidget`
-`visual`, `visual`, `Visual`
-`width`, `width`, `Width`

DESCRIPTION
WIDGET COMMAND

`pathName cget option`
`pathName configure ?option? ?value option value ...?`
NAME

labelframe - Create and manipulate labelframe widgets

SYNOPSIS

labelframe pathName ?options?

STANDARD OPTIONS

-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text

WIDGET-SPECIFIC OPTIONS

Command-Line Name: -background
Database Name: background
Database Class: Background

This option is the same as the standard background option except that its value may also be specified as an empty string. In this case, the widget will display no background or border, and no colors will be consumed from its colormap for its background and border.
Command-Line Name: -class
Database Name: class
Database Class: Class
   Specifies a class for the window. This class will be used when querying the option database for the window's other options, and it will also be used later for other purposes such as bindings. The class option may not be changed with the configure widget command.

Command-Line Name: -colormap
Database Name: colormap
Database Class: Colormap
   Specifies a colormap to use for the window. The value may be either new, in which case a new colormap is created for the window and its children, or the name of another window (which must be on the same screen and have the same visual as pathName), in which case the new window will use the colormap from the specified window. If the colormap option is not specified, the new window uses the same colormap as its parent. This option may not be changed with the configure widget command.

Command-Line Name: -container
Database Name: container
Database Class: Container
   The value must be a boolean. If true, it means that this window will be used as a container in which some other application will be embedded (for example, a Tk toplevel can be embedded using the -use option). The window will support the appropriate window manager protocols for things like geometry requests. The window should not have any children of its own in this application. This option may not be changed with the configure widget command.

Command-Line Name: -height
Database Name: height
Database Class: Height
   Specifies the desired height for the window in any of the forms acceptable to Tk_GetPixels. If this option is less than or equal to zero then the window will not request any size at all.
Command-Line Name: -labelanchor
Database Name: labelAnchor
Database Class: LabelAnchor
   Specifies where to place the label. A label is only displayed if the -text option is not the empty string. Valid values for this option are (listing them clockwise) nw, n, ne, en, e, es, se, s, sw, ws, w and wn. The default value is nw.

Command-Line Name: -labelwidget
Database Name: labelWidget
Database Class: LabelWidget
   Specifies a widget to use as label. This overrides any -text option. The widget must exist before being used as -labelwidget and if it is not a descendant of this window, it will be raised above it in the stacking order.

Command-Line Name: -visual
Database Name: visual
Database Class: Visual
   Specifies visual information for the new window in any of the forms accepted by Tk_GetVisual. If this option is not specified, the new window will use the same visual as its parent. The visual option may not be modified with the configure widget command.

Command-Line Name: -width
Database Name: width
Database Class: Width
   Specifies the desired width for the window in any of the forms acceptable to Tk_GetPixels. If this option is less than or equal to zero then the window will not request any size at all.

DESCRIPTION

The labelframe command creates a new window (given by the pathName argument) and makes it into a labelframe widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the labelframe such as its
background color and relief. The `labelframe` command returns the path name of the new window.

A labelframe is a simple widget. Its primary purpose is to act as a spacer or container for complex window layouts. It has the features of a `frame` plus the ability to display a label.

---

### WIDGET COMMAND

The `labelframe` command creates a new Tcl command whose name is the same as the path name of the labelframe's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*PathName* is the name of the command, which is the same as the labelframe widget's path name. *Option* and the *args* determine the exact behavior of the command. The following commands are possible for frame widgets:

**pathName cget option**

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the `labelframe` command.

**pathName configure ?option? ?value option value ...?**

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see [Tk_ConfigureInfo](#) for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option-value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string.
Option may have any of the values accepted by the labelframe command.

**BINDINGS**

When a new labelframe is created, it has no default event bindings: labelframes are not intended to be interactive.

**EXAMPLE**

This shows how to build part of a GUI for a hamburger vendor. The labelframe widgets are used to organize the available choices by the kinds of things that the choices are being made over.

```plaintext
grid [labelframe .burger -text "Burger"] \ 
[labelframe .bun -text "Bun"] -sticky news
grid [labelframe .cheese -text "Cheese Option"] \ 
[labelframe .pickle -text "Pickle Option"] -sticky news

foreach {type name val} {
    burger Beef   beef
    burger Lamb   lamb
    burger Vegetarian beans

    bun       Plain  white
    bun       Sesame seeds
    bun       Wholemeal brown

    cheese None    none
    cheese Cheddar cheddar
    cheese Edam   edam
    cheese Brie   brie
    cheese Gruy\u00e8re gruyere
    cheese "Monterey Jack" jack

    pickle None    none
    pickle Gherkins gherkins
    pickle Onions  onion
```
pickle Chili  chili
}

set w [radiobutton .$type.$val -text $name -anchor $type -variable $type -value $val]
pack $w -side top -fill x
}

set burger beef
set bun white
set cheese none
set pickle none

SEE ALSO

frame, label, ttk::labelframe

KEYWORDS

labelframe, widget

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
    scale - Create and manipulate scale widgets

SYNOPSIS

STANDARD OPTIONS
    -activebackground, activeBackground, Foreground
    -background or -bg, background, Background
    -borderwidth or -bd, borderWidth, BorderWidth
    -cursor, cursor, Cursor
    -font, font, Font
    -foreground or -fg, foreground, Foreground
    -highlightbackground, highlightBackground, HighlightBackground
    -highlightcolor, highlightColor, HighlightColor
    -highlightthickness, highlightThickness, HighlightThickness
    -orient, orient, Orient
    -relief, relief, Relief
    -repeatdelay, repeatDelay, RepeatDelay
    -repeatinterval, repeatInterval, RepeatInterval
    -takefocus, takeFocus, TakeFocus
    -troughcolor, troughColor, Background

WIDGET-SPECIFIC OPTIONS
    -bigincrement, bigIncrement, BigIncrement
    -command, command, Command
    -digits, digits, Digits
    -from, from, From
    -label, label, Label
    -length, length, Length
    -resolution, resolution, Resolution
    -showvalue, showValue, ShowValue
    -sliderlength, sliderLength, SliderLength
    -sliderrelief, sliderRelief, SliderRelief
-state, state, State
-tickinterval, tickInterval, TickInterval
-to, to, To
-variable, variable, Variable
-width, width, Width

**DESCRIPTION**

**WIDGET COMMAND**

`pathName cget option`
`pathName configure ?option? ?value option value ...`?
`pathName coords ?value?`
`pathName get ?x y?`
`pathName identify x y`
`pathName set value`

**BINDINGS**

**KEYWORDS**

### NAME

scale - Create and manipulate scale widgets

### SYNOPSIS

`scale pathName ?options?`

### STANDARD OPTIONS

-`-activebackground`, `activeBackground`, `Foreground`
-`-background or -bg`, `background`, `Background`
-`-borderwidth or -bd`, `borderWidth`, `BorderWidth`
-`-cursor`, `cursor`, `Cursor`
-`-font`, `font`, `Font`
-`-foreground or -fg`, `foreground`, `Foreground`
-`-highlightbackground`, `highlightBackground`, `HighlightBackground`
-`-highlightcolor`, `highlightColor`, `HighlightColor`
-`-highlightthickness`, `highlightThickness`, `HighlightThickness`
-`-orient`, `orient`, `Orient`
-`-relief`, `relief`, `Relief`
-`-repeatdelay`, `repeatDelay`, `RepeatDelay`
**WIDGET-SPECIFIC OPTIONS**

| Command-Line Name: **-bigincrement** |
| Database Name: **bigIncrement**  |
| Database Class: **BigIncrement**  |
| Some interactions with the scale cause its value to change by “large” increments; this option specifies the size of the large increments. If specified as 0, the large increments default to 1/10 the range of the scale. |

| Command-Line Name: **-command** |
| Database Name: **command**  |
| Database Class: **Command**  |
| Specifies the prefix of a Tcl command to invoke whenever the scale's value is changed via a widget command. The actual command consists of this option followed by a space and a real number indicating the new value of the scale. |

| Command-Line Name: **-digits** |
| Database Name: **digits**  |
| Database Class: **Digits**  |
| An integer specifying how many significant digits should be retained when converting the value of the scale to a string. If the number is less than or equal to zero, then the scale picks the smallest value that guarantees that every possible slider position prints as a different string. |

| Command-Line Name: **-from** |
| Database Name: **from**  |
| Database Class: **From**  |
| A real value corresponding to the left or top end of the scale. |

| Command-Line Name: **-label** |
| Database Name: **label**  |
Database Class: **Label**
A string to display as a label for the scale. For vertical scales the label is displayed just to the right of the top end of the scale. For horizontal scales the label is displayed just above the left end of the scale. If the option is specified as an empty string, no label is displayed.

Command-Line Name: **-length**
Database Name: **length**
Database Class: **Length**
   Specifies the desired long dimension of the scale in screen units (i.e. any of the forms acceptable to **Tk.GetPixels**). For vertical scales this is the scale's height; for horizontal scales it is the scale's width.

Command-Line Name: **-resolution**
Database Name: **resolution**
Database Class: **Resolution**
   A real value specifying the resolution for the scale. If this value is greater than zero then the scale's value will always be rounded to an even multiple of this value, as will tick marks and the endpoints of the scale. If the value is less than zero then no rounding occurs. Defaults to 1 (i.e., the value will be integral).

Command-Line Name: **-showvalue**
Database Name: **showValue**
Database Class: **ShowValue**
   Specifies a boolean value indicating whether or not the current value of the scale is to be displayed.

Command-Line Name: **-sliderlength**
Database Name: **sliderLength**
Database Class: **SliderLength**
   Specifies the size of the slider, measured in screen units along the slider's long dimension. The value may be specified in any of the forms acceptable to **Tk.GetPixels**.

Command-Line Name: **-sliderrelief**
Database Name: sliderRelief
Database Class: SliderRelief
   Specifies the relief to use when drawing the slider, such as raised or sunken.

Command-Line Name: -state
Database Name: state
Database Class: State
   Specifies one of three states for the scale: normal, active, or disabled. If the scale is disabled then the value may not be changed and the scale will not activate. If the scale is active, the slider is displayed using the color specified by the activeBackground option.

Command-Line Name: -tickinterval
Database Name: tickInterval
Database Class: TickInterval
   Must be a real value. Determines the spacing between numerical tick marks displayed below or to the left of the slider. If 0, no tick marks will be displayed.

Command-Line Name: -to
Database Name: to
Database Class: To
   Specifies a real value corresponding to the right or bottom end of the scale. This value may be either less than or greater than the from option.

Command-Line Name: -variable
Database Name: variable
Database Class: Variable
   Specifies the name of a global variable to link to the scale. Whenever the value of the variable changes, the scale will update to reflect this value. Whenever the scale is manipulated interactively, the variable will be modified to reflect the scale's new value.

Command-Line Name: -width
Database Name: **width**  
Database Class: **Width**  
   Specifies the desired narrow dimension of the trough in screen units (i.e. any of the forms acceptable to **Tk_GetPixels**). For vertical scales this is the trough's width; for horizontal scales this is the trough's height.

---

**DESCRIPTION**

The **scale** command creates a new window (given by the **pathName** argument) and makes it into a scale widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the scale such as its colors, orientation, and relief. The **scale** command returns its **pathName** argument. At the time this command is invoked, there must not exist a window named **pathName**, but **pathName**'s parent must exist.

A scale is a widget that displays a rectangular **trough** and a small **slider**. The trough corresponds to a range of real values (determined by the **from**, **to**, and **resolution** options), and the position of the slider selects a particular real value. The slider’s position (and hence the scale's value) may be adjusted with the mouse or keyboard as described in the **BINDINGS** section below. Whenever the scale's value is changed, a Tcl command is invoked (using the **command** option) to notify other interested widgets of the change. In addition, the value of the scale can be linked to a Tcl variable (using the **variable** option), so that changes in either are reflected in the other.

Three annotations may be displayed in a scale widget: a label appearing at the top right of the widget (top left for horizontal scales), a number displayed just to the left of the slider (just above the slider for horizontal scales), and a collection of numerical tick marks just to the left of the current value (just below the trough for horizontal scales). Each of these three annotations may be enabled or disabled using the configuration options.

---

**WIDGET COMMAND**

---
The `scale` command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for scale widgets:

`pathName cget option`

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the `scale` command.

`pathName configure ?option? ?value option value ...?`

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for `pathName` (see `Tk_ConfigureInfo` for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option-value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the `scale` command.

`pathName coords ?value?`

Returns a list whose elements are the x and y coordinates of the point along the centerline of the trough that corresponds to *value*. If *value* is omitted then the scale's current value is used.

`pathName get ?x y?`

If *x* and *y* are omitted, returns the current value of the scale. If *x* and *y* are specified, they give pixel coordinates within the widget; the command returns the scale value corresponding to the given
Only one of $x$ or $y$ is used: for horizontal scales $y$ is ignored, and for vertical scales $x$ is ignored.

**pathName identify $x$ $y**

Returns a string indicating what part of the scale lies under the coordinates given by $x$ and $y$. A return value of **slider** means that the point is over the slider; **trough1** means that the point is over the portion of the slider above or to the left of the slider; and **trough2** means that the point is over the portion of the slider below or to the right of the slider. If the point is not over one of these elements, an empty string is returned.

**pathName set value**

This command is invoked to change the current value of the scale, and hence the position at which the slider is displayed. **Value** gives the new value for the scale. The command has no effect if the scale is disabled.

**BINDINGS**

Tk automatically creates class bindings for scales that give them the following default behavior. Where the behavior is different for vertical and horizontal scales, the horizontal behavior is described in parentheses.

[1]

If button 1 is pressed in the trough, the scale's value will be incremented or decremented by the value of the **resolution** option so that the slider moves in the direction of the cursor. If the button is held down, the action auto-repeats.

[2]

If button 1 is pressed over the slider, the slider can be dragged with the mouse.

[3]

If button 1 is pressed in the trough with the Control key down, the slider moves all the way to the end of its range, in the direction
towards the mouse cursor.

[4] If button 2 is pressed, the scale's value is set to the mouse position. If the mouse is dragged with button 2 down, the scale's value changes with the drag.

[5] The Up and Left keys move the slider up (left) by the value of the resolution option.

[6] The Down and Right keys move the slider down (right) by the value of the resolution option.

[7] Control-Up and Control-Left move the slider up (left) by the value of the bigIncrement option.

[8] Control-Down and Control-Right move the slider down (right) by the value of the bigIncrement option.

[9] Home moves the slider to the top (left) end of its range.

[10] End moves the slider to the bottom (right) end of its range.

If the scale is disabled using the state option then none of the above bindings have any effect.

The behavior of scales can be changed by defining new bindings for individual widgets or by redefining the class bindings.

**KEYWORDS**

scale, slider, trough, widget
NAME

tk::combobox - text field with popdown selection list

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS

-class

-cursor, cursor, Cursor

-style

-takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS

-exportselection, exportSelection, ExportSelection

-justify, justify, Justify

-height, height, Height

-postcommand, postCommand, PostCommand

-state, state, State

-textvariable, textVariable, TextVariable

-values, values, Values

-width, width, Width

WIDGET COMMAND

pathName cget option

pathName configure ?option? ?value option value ...?

pathName current ?newIndex?

pathName get

pathName identify x y

pathName instate stateSpec ?script?

pathName set value

pathName state ?stateSpec?

VIRTUAL EVENTS

SEE ALSO

KEYWORDS
ttk::combobox - text field with popdown selection list

ttk::combobox *pathName* ?*options*?

A *ttk::combobox* combines a text field with a pop-down list of values; the user may select the value of the text field from among the values in the list.

**STANDARD OPTIONS**

- *-class*
- *-cursor, cursor, Cursor*
- *-style*
- *-takefocus, takeFocus, TakeFocus*

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: *-exportselection*
Database Name: *exportSelection*
Database Class: *ExportSelection*
   Boolean value. If set, the widget selection is linked to the X selection.

Command-Line Name: *-justify*
Database Name: *justify*
Database Class: *Justify*
   Specifies how the text is aligned within the widget. One of *left*, *center*, or *right*.

Command-Line Name: *-height*
Database Name: *height*
Database Class: *Height*
Specifies the height of the pop-down listbox, in rows.

Command-Line Name: **-postcommand**
Database Name: **postCommand**
Database Class: **PostCommand**

A Tcl script to evaluate immediately before displaying the listbox. The **-postcommand** script may specify the **-values** to display.

Command-Line Name: **-state**
Database Name: **state**
Database Class: **State**

One of **normal**, **readonly**, or **disabled**. In the **readonly** state, the value may not be edited directly, and the user can only select one of the **-values** from the dropdown list. In the **normal** state, the text field is directly editable. In the **disabled** state, no interaction is possible.

Command-Line Name: **-textvariable**
Database Name: **textVariable**
Database Class: **TextVariable**

Specifies the name of a variable whose value is linked to the widget value. Whenever the variable changes value the widget value is updated, and vice versa.

Command-Line Name: **-values**
Database Name: **values**
Database Class: **Values**

Specifies the list of values to display in the drop-down listbox.

Command-Line Name: **-width**
Database Name: **width**
Database Class: **Width**

Specifies an integer value indicating the desired width of the entry window, in average-size characters of the widget's font.

**WIDGET COMMAND**

*pathName cget option*
Returns the current value of the specified option. See \texttt{ttk::widget(n)}.

\texttt{pathName configure \?option\? \?value option value \ldots\?}
Modify or query widget options. See \texttt{ttk::widget(n)}.

\texttt{pathName current \?newIndex?}
If \texttt{newIndex} is supplied, sets the combobox value to the element at position \texttt{newIndex} in the list of \texttt{-values}. Otherwise, returns the index of the current value in the list of \texttt{-values} or \texttt{-1} if the current value does not appear in the list.

\texttt{pathName get}
Returns the current value of the combobox.

\texttt{pathName identify x y}
Returns the name of the element at position \texttt{x}, \texttt{y}. See \texttt{ttk::widget(n)}.

\texttt{pathName instate stateSpec \?script?}
Test the widget state. See \texttt{ttk::widget(n)}.

\texttt{pathName set value}
Sets the value of the combobox to \texttt{value}.

\texttt{pathName state \?stateSpec?}
Modify or query the widget state. See \texttt{ttk::widget(n)}.

The combobox widget also supports the following \texttt{ttk::entry} widget commands (see \texttt{ttk::entry(n)} for details):

- \texttt{bbox}
- \texttt{delete}
- \texttt{icursor}

- \texttt{index}
- \texttt{insert}
- \texttt{selection}

- \texttt{xview}
VIRTUAL EVENTS

The combobox widget generates a \texttt{<<ComboboxSelected>>} virtual event when the user selects an element from the list of values. If the selection action unposts the listbox, this event is delivered after the listbox is unposted.

SEE ALSO

ttk::widget, ttk::entry

KEYWORDS

choice, entry, list box, text box, widget
NAME

ttk::style - Manipulate style database

SYNOPSIS

SEE ALSO

KEYWORDS
NOTES


DEFINITIONS

Each widget is assigned a style, which specifies the set of elements making up the widget and how they are arranged, along with dynamic and default settings for element options. By default, the style name is the same as the widget's class; this may be overridden by the -style option.

A theme is a collection of elements and styles which controls the overall look and feel of an application.

DESCRIPTION

The ttk::style command takes the following arguments:

ttk::style configure style ?-option ?value option value...? ?
Sets the default value of the specified option(s) in style.

ttk::style map style ?-option { statespec value... }?
Sets dynamic values of the specified option(s) in style. Each statespec / value pair is examined in order; the value corresponding to the first matching statespec is used.

ttk::style lookup style -option ?state ?default??
Returns the value specified for -option in style style in state state, using the standard lookup rules for element options. state is a list of state names; if omitted, it defaults to all bits off (the “normal” state). If the default argument is present, it is used as a fallback value in case no specification for -option is found.

ttk::style layout style ?layoutSpec?
Define the widget layout for style style. See LAYOUTS below for the format of layoutSpec. If layoutSpec is omitted, return the layout
specification for style *style*.

**ttk::style element create** *elementName* *type* *?args...*?

Creates a new element in the current theme of type *type*. The only cross-platform built-in element type is *image* (see *ttk_image*(n)) but themes may define other element types (see *Ttk_RegisterElementFactory*). On suitable versions of Windows an element factory is registered to create Windows theme elements (see *ttk_vsapi*(n)).

**ttk::style element names**

Returns the list of elements defined in the current theme.

**ttk::style element options** *element*

Returns the list of *element*'s options.

**ttk::style theme create** *themeName* *?-parent* *basedon*? *?-settings* *script...*?

Creates a new theme. It is an error if *themeName* already exists. If *-parent* is specified, the new theme will inherit styles, elements, and layouts from the parent theme *basedon*. If *-settings* is present, *script* is evaluated in the context of the new theme as per *ttk::style theme settings*.

**ttk::style theme settings** *themeName* *script*

Temporarily sets the current theme to *themeName*, evaluate *script*, then restore the previous theme. Typically *script* simply defines styles and elements, though arbitrary Tcl code may appear.

**ttk::style theme names**

Returns a list of all known themes.

**ttk::style theme use** *themeName*

Sets the current theme to *themeName*, and refreshes all widgets.

**LAYOUTS**

A *layout* specifies a list of elements, each followed by one or more
options specifying how to arrange the element. The layout mechanism uses a simplified version of the \texttt{pack} geometry manager: given an initial cavity, each element is allocated a parcel. Valid options are:

\textbf{\texttt{-side} \textit{side}}

Specifies which side of the cavity to place the element; one of \texttt{left}, \texttt{right}, \texttt{top}, or \texttt{bottom}. If omitted, the element occupies the entire cavity.

\textbf{\texttt{-sticky \{nswe\}}}

Specifies where the element is placed inside its allocated parcel.

\textbf{\texttt{-children \{ sublayout\ldots\} }}

Specifies a list of elements to place inside the element.

For example:

\begin{verbatim}
 ttk::style layout Horizontal.TScrollbar {
   Scrollbar.trough -children {
     Scrollbar.leftarrow -side left
     Scrollbar.rightarrow -side right
     Horizontal.Scrollbar.thumb -side left -sticky
   }
}
\end{verbatim}

\textbf{SEE ALSO}

\texttt{ttk::intro, ttk::widget, photo, ttk_image}

\textbf{KEYWORDS}

\texttt{style, theme, appearance}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2004 Joe English
colors - symbolic color names recognized by Tk

**NAME**

**DESCRIPTION**

Tk recognizes many symbolic color names (e.g., **red**) when specifying colors. The symbolic names recognized by Tk and their 8-bit-per-channel RGB values are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice blue</td>
<td>240</td>
<td>248</td>
<td>255</td>
</tr>
<tr>
<td>AliceBlue</td>
<td>240</td>
<td>248</td>
<td>255</td>
</tr>
<tr>
<td>antique white</td>
<td>250</td>
<td>235</td>
<td>215</td>
</tr>
<tr>
<td>AntiqueWhite</td>
<td>250</td>
<td>235</td>
<td>215</td>
</tr>
<tr>
<td>AntiqueWhite1</td>
<td>255</td>
<td>239</td>
<td>219</td>
</tr>
<tr>
<td>AntiqueWhite2</td>
<td>238</td>
<td>223</td>
<td>204</td>
</tr>
<tr>
<td>AntiqueWhite3</td>
<td>205</td>
<td>192</td>
<td>176</td>
</tr>
<tr>
<td>Color</td>
<td>RGB1</td>
<td>RGB2</td>
<td>RGB3</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>AntiqueWhite4</td>
<td>139</td>
<td>131</td>
<td>120</td>
</tr>
<tr>
<td>aquamarine</td>
<td>127</td>
<td>255</td>
<td>212</td>
</tr>
<tr>
<td>aquamarine1</td>
<td>127</td>
<td>255</td>
<td>212</td>
</tr>
<tr>
<td>aquamarine2</td>
<td>118</td>
<td>238</td>
<td>198</td>
</tr>
<tr>
<td>aquamarine3</td>
<td>102</td>
<td>205</td>
<td>170</td>
</tr>
<tr>
<td>aquamarine4</td>
<td>69</td>
<td>139</td>
<td>116</td>
</tr>
<tr>
<td>azure</td>
<td>240</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>azure1</td>
<td>240</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>azure2</td>
<td>224</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>azure3</td>
<td>193</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>azure4</td>
<td>131</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>beige</td>
<td>245</td>
<td>245</td>
<td>220</td>
</tr>
<tr>
<td>bisque</td>
<td>255</td>
<td>228</td>
<td>196</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>bisque1</td>
<td>255</td>
<td>228</td>
<td>196</td>
</tr>
<tr>
<td>bisque2</td>
<td>238</td>
<td>213</td>
<td>183</td>
</tr>
<tr>
<td>bisque3</td>
<td>205</td>
<td>183</td>
<td>158</td>
</tr>
<tr>
<td>bisque4</td>
<td>139</td>
<td>125</td>
<td>107</td>
</tr>
<tr>
<td>black</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>blanched almond</td>
<td>255</td>
<td>235</td>
<td>205</td>
</tr>
<tr>
<td>BlanchedAlmond</td>
<td>255</td>
<td>235</td>
<td>205</td>
</tr>
<tr>
<td>blue</td>
<td>0</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>blue violet</td>
<td>138</td>
<td>43</td>
<td>226</td>
</tr>
<tr>
<td>blue1</td>
<td>0</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>blue2</td>
<td>0</td>
<td>0</td>
<td>238</td>
</tr>
<tr>
<td>blue3</td>
<td>0</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>blue4</td>
<td>0</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>-------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>BlueViolet</td>
<td>138</td>
<td>43</td>
<td>226</td>
</tr>
<tr>
<td>brown</td>
<td>165</td>
<td>42</td>
<td>42</td>
</tr>
<tr>
<td>brown1</td>
<td>255</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>brown2</td>
<td>238</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>brown3</td>
<td>205</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>brown4</td>
<td>139</td>
<td>35</td>
<td>35</td>
</tr>
<tr>
<td>burlywood</td>
<td>222</td>
<td>184</td>
<td>135</td>
</tr>
<tr>
<td>burlywood1</td>
<td>255</td>
<td>211</td>
<td>155</td>
</tr>
<tr>
<td>burlywood2</td>
<td>238</td>
<td>197</td>
<td>145</td>
</tr>
<tr>
<td>burlywood3</td>
<td>205</td>
<td>170</td>
<td>125</td>
</tr>
<tr>
<td>burlywood4</td>
<td>139</td>
<td>115</td>
<td>85</td>
</tr>
<tr>
<td>cadet blue</td>
<td>95</td>
<td>158</td>
<td>160</td>
</tr>
<tr>
<td>CadetBlue</td>
<td>95</td>
<td>158</td>
<td>160</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>---------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>CadetBlue1</td>
<td>152</td>
<td>245</td>
<td>255</td>
</tr>
<tr>
<td>CadetBlue2</td>
<td>142</td>
<td>229</td>
<td>238</td>
</tr>
<tr>
<td>CadetBlue3</td>
<td>122</td>
<td>197</td>
<td>205</td>
</tr>
<tr>
<td>CadetBlue4</td>
<td>83</td>
<td>134</td>
<td>139</td>
</tr>
<tr>
<td>chartreuse</td>
<td>127</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>chartreuse1</td>
<td>127</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>chartreuse2</td>
<td>118</td>
<td>238</td>
<td>0</td>
</tr>
<tr>
<td>chartreuse3</td>
<td>102</td>
<td>205</td>
<td>0</td>
</tr>
<tr>
<td>chartreuse4</td>
<td>69</td>
<td>139</td>
<td>0</td>
</tr>
<tr>
<td>chocolate</td>
<td>210</td>
<td>105</td>
<td>30</td>
</tr>
<tr>
<td>chocolate1</td>
<td>255</td>
<td>127</td>
<td>36</td>
</tr>
<tr>
<td>chocolate2</td>
<td>238</td>
<td>118</td>
<td>33</td>
</tr>
<tr>
<td>chocolate3</td>
<td>205</td>
<td>102</td>
<td>29</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>------------------------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>chocolate4</td>
<td>139</td>
<td>69</td>
<td>19</td>
</tr>
<tr>
<td>coral</td>
<td>255</td>
<td>127</td>
<td>80</td>
</tr>
<tr>
<td>coral1</td>
<td>255</td>
<td>114</td>
<td>86</td>
</tr>
<tr>
<td>coral2</td>
<td>238</td>
<td>106</td>
<td>80</td>
</tr>
<tr>
<td>coral3</td>
<td>205</td>
<td>91</td>
<td>69</td>
</tr>
<tr>
<td>coral4</td>
<td>139</td>
<td>62</td>
<td>47</td>
</tr>
<tr>
<td>cornflower blue</td>
<td>100</td>
<td>149</td>
<td>237</td>
</tr>
<tr>
<td>CornflowerBlue</td>
<td>100</td>
<td>149</td>
<td>237</td>
</tr>
<tr>
<td>cornsilk</td>
<td>255</td>
<td>248</td>
<td>220</td>
</tr>
<tr>
<td>cornsilk1</td>
<td>255</td>
<td>248</td>
<td>220</td>
</tr>
<tr>
<td>cornsilk2</td>
<td>238</td>
<td>232</td>
<td>205</td>
</tr>
<tr>
<td>cornsilk3</td>
<td>205</td>
<td>200</td>
<td>177</td>
</tr>
<tr>
<td>cornsilk4</td>
<td>139</td>
<td>136</td>
<td>120</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>---------------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>cyan</td>
<td>0</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>cyan1</td>
<td>0</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>cyan2</td>
<td>0</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>cyan3</td>
<td>0</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>cyan4</td>
<td>0</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>dark blue</td>
<td>0</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>dark cyan</td>
<td>0</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>dark goldenrod</td>
<td>184</td>
<td>134</td>
<td>11</td>
</tr>
<tr>
<td>dark gray</td>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>dark green</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>dark grey</td>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>dark khaki</td>
<td>189</td>
<td>183</td>
<td>107</td>
</tr>
<tr>
<td>dark magenta</td>
<td>139</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>Color Description</td>
<td>RGB Values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark olive green</td>
<td>85, 107, 47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark orange</td>
<td>255, 140, 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark orchid</td>
<td>153, 50, 204</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark red</td>
<td>139, 0, 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark salmon</td>
<td>233, 150, 122</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark sea green</td>
<td>143, 188, 143</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark slate blue</td>
<td>72, 61, 139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark slate gray</td>
<td>47, 79, 79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark slate grey</td>
<td>47, 79, 79</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark turquoise</td>
<td>0, 206, 209</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dark violet</td>
<td>148, 0, 211</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DarkBlue</td>
<td>0, 0, 139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DarkCyan</td>
<td>0, 139, 139</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
</tr>
<tr>
<td>DarkGoldenrod</td>
<td>184</td>
<td>134</td>
<td>11</td>
</tr>
<tr>
<td>DarkGoldenrod1</td>
<td>255</td>
<td>185</td>
<td>15</td>
</tr>
<tr>
<td>DarkGoldenrod2</td>
<td>238</td>
<td>173</td>
<td>14</td>
</tr>
<tr>
<td>DarkGoldenrod3</td>
<td>205</td>
<td>149</td>
<td>12</td>
</tr>
<tr>
<td>DarkGoldenrod4</td>
<td>139</td>
<td>101</td>
<td>8</td>
</tr>
<tr>
<td>DarkGray</td>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>DarkGreen</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>DarkGrey</td>
<td>169</td>
<td>169</td>
<td>169</td>
</tr>
<tr>
<td>DarkKhaki</td>
<td>189</td>
<td>183</td>
<td>107</td>
</tr>
<tr>
<td>DarkMagenta</td>
<td>139</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>DarkOliveGreen</td>
<td>85</td>
<td>107</td>
<td>47</td>
</tr>
<tr>
<td>DarkOliveGreen1</td>
<td>202</td>
<td>255</td>
<td>112</td>
</tr>
<tr>
<td>DarkOliveGreen2</td>
<td>188</td>
<td>238</td>
<td>104</td>
</tr>
<tr>
<td>Color Name</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>DarkOliveGreen3</td>
<td>162</td>
<td>205</td>
<td>90</td>
</tr>
<tr>
<td>DarkOliveGreen4</td>
<td>110</td>
<td>139</td>
<td>61</td>
</tr>
<tr>
<td>DarkOrange</td>
<td>255</td>
<td>140</td>
<td>0</td>
</tr>
<tr>
<td>DarkOrange1</td>
<td>255</td>
<td>127</td>
<td>0</td>
</tr>
<tr>
<td>DarkOrange2</td>
<td>238</td>
<td>118</td>
<td>0</td>
</tr>
<tr>
<td>DarkOrange3</td>
<td>205</td>
<td>102</td>
<td>0</td>
</tr>
<tr>
<td>DarkOrange4</td>
<td>139</td>
<td>69</td>
<td>0</td>
</tr>
<tr>
<td>DarkOrchid</td>
<td>153</td>
<td>50</td>
<td>204</td>
</tr>
<tr>
<td>DarkOrchid1</td>
<td>191</td>
<td>62</td>
<td>255</td>
</tr>
<tr>
<td>DarkOrchid2</td>
<td>178</td>
<td>58</td>
<td>238</td>
</tr>
<tr>
<td>DarkOrchid3</td>
<td>154</td>
<td>50</td>
<td>205</td>
</tr>
<tr>
<td>DarkOrchid4</td>
<td>104</td>
<td>34</td>
<td>139</td>
</tr>
<tr>
<td>DarkRed</td>
<td>139</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Color Name</td>
<td>Red Value</td>
<td>Green Value</td>
<td>Blue Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>-------------</td>
<td>------------</td>
</tr>
<tr>
<td>DarkSalmon</td>
<td>233</td>
<td>150</td>
<td>122</td>
</tr>
<tr>
<td>DarkSeaGreen</td>
<td>143</td>
<td>188</td>
<td>143</td>
</tr>
<tr>
<td>DarkSeaGreen1</td>
<td>193</td>
<td>255</td>
<td>193</td>
</tr>
<tr>
<td>DarkSeaGreen2</td>
<td>180</td>
<td>238</td>
<td>180</td>
</tr>
<tr>
<td>DarkSeaGreen3</td>
<td>155</td>
<td>205</td>
<td>155</td>
</tr>
<tr>
<td>DarkSeaGreen4</td>
<td>105</td>
<td>139</td>
<td>105</td>
</tr>
<tr>
<td>DarkSlateBlue</td>
<td>72</td>
<td>61</td>
<td>139</td>
</tr>
<tr>
<td>DarkSlateGray</td>
<td>47</td>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>DarkSlateGray1</td>
<td>151</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>DarkSlateGray2</td>
<td>141</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>DarkSlateGray3</td>
<td>121</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>DarkSlateGray4</td>
<td>82</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>DarkSlateGrey</td>
<td>47</td>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>DarkTurquoise</td>
<td>0</td>
<td>206</td>
<td>209</td>
</tr>
<tr>
<td>DarkViolet</td>
<td>148</td>
<td>0</td>
<td>211</td>
</tr>
<tr>
<td>deep pink</td>
<td>255</td>
<td>20</td>
<td>147</td>
</tr>
<tr>
<td>deep sky blue</td>
<td>0</td>
<td>191</td>
<td>255</td>
</tr>
<tr>
<td>DeepPink</td>
<td>255</td>
<td>20</td>
<td>147</td>
</tr>
<tr>
<td>DeepPink1</td>
<td>255</td>
<td>20</td>
<td>147</td>
</tr>
<tr>
<td>DeepPink2</td>
<td>238</td>
<td>18</td>
<td>137</td>
</tr>
<tr>
<td>DeepPink3</td>
<td>205</td>
<td>16</td>
<td>118</td>
</tr>
<tr>
<td>DeepPink4</td>
<td>139</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>DeepSkyBlue</td>
<td>0</td>
<td>191</td>
<td>255</td>
</tr>
<tr>
<td>DeepSkyBlue1</td>
<td>0</td>
<td>191</td>
<td>255</td>
</tr>
<tr>
<td>DeepSkyBlue2</td>
<td>0</td>
<td>178</td>
<td>238</td>
</tr>
<tr>
<td>DeepSkyBlue3</td>
<td>0</td>
<td>154</td>
<td>205</td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>-------------------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>DeepSkyBlue4</td>
<td>0</td>
<td>104</td>
<td>139</td>
</tr>
<tr>
<td>dim gray</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>dim grey</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>DimGray</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>DimGrey</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>dodger blue</td>
<td>30</td>
<td>144</td>
<td>255</td>
</tr>
<tr>
<td>DodgerBlue</td>
<td>30</td>
<td>144</td>
<td>255</td>
</tr>
<tr>
<td>DodgerBlue1</td>
<td>30</td>
<td>144</td>
<td>255</td>
</tr>
<tr>
<td>DodgerBlue2</td>
<td>28</td>
<td>134</td>
<td>238</td>
</tr>
<tr>
<td>DodgerBlue3</td>
<td>24</td>
<td>116</td>
<td>205</td>
</tr>
<tr>
<td>DodgerBlue4</td>
<td>16</td>
<td>78</td>
<td>139</td>
</tr>
<tr>
<td>firebrick</td>
<td>178</td>
<td>34</td>
<td>34</td>
</tr>
<tr>
<td>firebrick1</td>
<td>255</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>Color</td>
<td>RGB 1</td>
<td>RGB 2</td>
<td>RGB 3</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>firebrick2</td>
<td>238</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>firebrick3</td>
<td>205</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>firebrick4</td>
<td>139</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>floral white</td>
<td>255</td>
<td>250</td>
<td>240</td>
</tr>
<tr>
<td>FloralWhite</td>
<td>255</td>
<td>250</td>
<td>240</td>
</tr>
<tr>
<td>forest green</td>
<td>34</td>
<td>139</td>
<td>34</td>
</tr>
<tr>
<td>ForestGreen</td>
<td>34</td>
<td>139</td>
<td>34</td>
</tr>
<tr>
<td>gainsboro</td>
<td>220</td>
<td>220</td>
<td>220</td>
</tr>
<tr>
<td>ghost white</td>
<td>248</td>
<td>248</td>
<td>255</td>
</tr>
<tr>
<td>GhostWhite</td>
<td>248</td>
<td>248</td>
<td>255</td>
</tr>
<tr>
<td>gold</td>
<td>255</td>
<td>215</td>
<td>0</td>
</tr>
<tr>
<td>gold1</td>
<td>255</td>
<td>215</td>
<td>0</td>
</tr>
<tr>
<td>gold2</td>
<td>238</td>
<td>201</td>
<td>0</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>---------------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>gold3</td>
<td>205</td>
<td>173</td>
<td>0</td>
</tr>
<tr>
<td>gold4</td>
<td>139</td>
<td>117</td>
<td>0</td>
</tr>
<tr>
<td>goldenrod</td>
<td>218</td>
<td>165</td>
<td>32</td>
</tr>
<tr>
<td>goldenrod1</td>
<td>255</td>
<td>193</td>
<td>37</td>
</tr>
<tr>
<td>goldenrod2</td>
<td>238</td>
<td>180</td>
<td>34</td>
</tr>
<tr>
<td>goldenrod3</td>
<td>205</td>
<td>155</td>
<td>29</td>
</tr>
<tr>
<td>goldenrod4</td>
<td>139</td>
<td>105</td>
<td>20</td>
</tr>
<tr>
<td>gray</td>
<td>190</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>gray0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>gray1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>gray2</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>gray3</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>gray4</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>gray5</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>---------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>gray6</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>gray7</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>gray8</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>gray9</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>gray10</td>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>gray11</td>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>gray12</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>gray13</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>gray14</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>gray15</td>
<td>38</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>gray16</td>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>gray17</td>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>gray18</td>
<td>46</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>gray19</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>gray20</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>gray21</td>
<td>54</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>gray22</td>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>gray23</td>
<td>59</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>gray24</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>gray25</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>gray26</td>
<td>66</td>
<td>66</td>
<td>66</td>
</tr>
<tr>
<td>gray27</td>
<td>69</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>gray28</td>
<td>71</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>gray29</td>
<td>74</td>
<td>74</td>
<td>74</td>
</tr>
<tr>
<td>gray30</td>
<td>77</td>
<td>77</td>
<td>77</td>
</tr>
<tr>
<td>gray31</td>
<td>79</td>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>gray32</td>
<td>82</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td>gray33</td>
<td>84</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>gray34</td>
<td>87</td>
<td>87</td>
<td>87</td>
</tr>
<tr>
<td>gray35</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>gray36</td>
<td>92</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>gray37</td>
<td>94</td>
<td>94</td>
<td>94</td>
</tr>
<tr>
<td>gray38</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>gray39</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>gray40</td>
<td>102</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>gray41</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>gray42</td>
<td>107</td>
<td>107</td>
<td>107</td>
</tr>
<tr>
<td>gray43</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>gray44</td>
<td>112</td>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>gray45</td>
<td>115</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>gray46</td>
<td>117</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>gray47</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>gray48</td>
<td>122</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>gray49</td>
<td>125</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>gray50</td>
<td>127</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>gray51</td>
<td>130</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>gray52</td>
<td>133</td>
<td>133</td>
<td>133</td>
</tr>
<tr>
<td>gray53</td>
<td>135</td>
<td>135</td>
<td>135</td>
</tr>
<tr>
<td>gray54</td>
<td>138</td>
<td>138</td>
<td>138</td>
</tr>
<tr>
<td>gray55</td>
<td>140</td>
<td>140</td>
<td>140</td>
</tr>
<tr>
<td>gray56</td>
<td>143</td>
<td>143</td>
<td>143</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>gray96</td>
<td>245</td>
<td>245</td>
<td>245</td>
</tr>
<tr>
<td>gray97</td>
<td>247</td>
<td>247</td>
<td>247</td>
</tr>
<tr>
<td>gray98</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>gray99</td>
<td>252</td>
<td>252</td>
<td>252</td>
</tr>
<tr>
<td>gray100</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>green</td>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>green yellow</td>
<td>173</td>
<td>255</td>
<td>47</td>
</tr>
<tr>
<td>green1</td>
<td>0</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>green2</td>
<td>0</td>
<td>238</td>
<td>0</td>
</tr>
<tr>
<td>green3</td>
<td>0</td>
<td>205</td>
<td>0</td>
</tr>
<tr>
<td>green4</td>
<td>0</td>
<td>139</td>
<td>0</td>
</tr>
<tr>
<td>GreenYellow</td>
<td>173</td>
<td>255</td>
<td>47</td>
</tr>
<tr>
<td>grey</td>
<td>190</td>
<td>190</td>
<td>190</td>
</tr>
<tr>
<td>grey0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-------</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>grey1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>grey2</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>grey3</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>grey4</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>grey5</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>grey6</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>grey7</td>
<td>18</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>grey8</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>grey9</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>grey10</td>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>grey11</td>
<td>28</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>grey12</td>
<td>31</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>grey13</td>
<td>33</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>grey14</td>
<td>36</td>
<td>36</td>
<td>36</td>
</tr>
<tr>
<td>grey15</td>
<td>38</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>grey16</td>
<td>41</td>
<td>41</td>
<td>41</td>
</tr>
<tr>
<td>grey17</td>
<td>43</td>
<td>43</td>
<td>43</td>
</tr>
<tr>
<td>grey18</td>
<td>46</td>
<td>46</td>
<td>46</td>
</tr>
<tr>
<td>grey19</td>
<td>48</td>
<td>48</td>
<td>48</td>
</tr>
<tr>
<td>grey20</td>
<td>51</td>
<td>51</td>
<td>51</td>
</tr>
<tr>
<td>grey21</td>
<td>54</td>
<td>54</td>
<td>54</td>
</tr>
<tr>
<td>grey22</td>
<td>56</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>grey23</td>
<td>59</td>
<td>59</td>
<td>59</td>
</tr>
<tr>
<td>grey24</td>
<td>61</td>
<td>61</td>
<td>61</td>
</tr>
<tr>
<td>grey25</td>
<td>64</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>grey26</td>
<td>66</td>
<td>66</td>
<td>66</td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>grey27</td>
<td>69</td>
<td>69</td>
<td>69</td>
</tr>
<tr>
<td>grey28</td>
<td>71</td>
<td>71</td>
<td>71</td>
</tr>
<tr>
<td>grey29</td>
<td>74</td>
<td>74</td>
<td>74</td>
</tr>
<tr>
<td>grey30</td>
<td>77</td>
<td>77</td>
<td>77</td>
</tr>
<tr>
<td>grey31</td>
<td>79</td>
<td>79</td>
<td>79</td>
</tr>
<tr>
<td>grey32</td>
<td>82</td>
<td>82</td>
<td>82</td>
</tr>
<tr>
<td>grey33</td>
<td>84</td>
<td>84</td>
<td>84</td>
</tr>
<tr>
<td>grey34</td>
<td>87</td>
<td>87</td>
<td>87</td>
</tr>
<tr>
<td>grey35</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>grey36</td>
<td>92</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>grey37</td>
<td>94</td>
<td>94</td>
<td>94</td>
</tr>
<tr>
<td>grey38</td>
<td>97</td>
<td>97</td>
<td>97</td>
</tr>
<tr>
<td>grey39</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>--------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>grey40</td>
<td>102</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>grey41</td>
<td>105</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>grey42</td>
<td>107</td>
<td>107</td>
<td>107</td>
</tr>
<tr>
<td>grey43</td>
<td>110</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>grey44</td>
<td>112</td>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>grey45</td>
<td>115</td>
<td>115</td>
<td>115</td>
</tr>
<tr>
<td>grey46</td>
<td>117</td>
<td>117</td>
<td>117</td>
</tr>
<tr>
<td>grey47</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>grey48</td>
<td>122</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>grey49</td>
<td>125</td>
<td>125</td>
<td>125</td>
</tr>
<tr>
<td>grey50</td>
<td>127</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>grey51</td>
<td>130</td>
<td>130</td>
<td>130</td>
</tr>
<tr>
<td>grey65</td>
<td>166</td>
<td>166</td>
<td>166</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>grey66</td>
<td>168</td>
<td>168</td>
<td>168</td>
</tr>
<tr>
<td>grey67</td>
<td>171</td>
<td>171</td>
<td>171</td>
</tr>
<tr>
<td>grey68</td>
<td>173</td>
<td>173</td>
<td>173</td>
</tr>
<tr>
<td>grey69</td>
<td>176</td>
<td>176</td>
<td>176</td>
</tr>
<tr>
<td>grey70</td>
<td>179</td>
<td>179</td>
<td>179</td>
</tr>
<tr>
<td>grey71</td>
<td>181</td>
<td>181</td>
<td>181</td>
</tr>
<tr>
<td>grey72</td>
<td>184</td>
<td>184</td>
<td>184</td>
</tr>
<tr>
<td>grey73</td>
<td>186</td>
<td>186</td>
<td>186</td>
</tr>
<tr>
<td>grey74</td>
<td>189</td>
<td>189</td>
<td>189</td>
</tr>
<tr>
<td>grey75</td>
<td>191</td>
<td>191</td>
<td>191</td>
</tr>
<tr>
<td>grey76</td>
<td>194</td>
<td>194</td>
<td>194</td>
</tr>
<tr>
<td>grey77</td>
<td>196</td>
<td>196</td>
<td>196</td>
</tr>
</tbody>
</table>
grey91  232  232  232
grey92  235  235  235
grey93  237  237  237
grey94  240  240  240
grey95  242  242  242
grey96  245  245  245
grey97  247  247  247
grey98  250  250  250
grey99  252  252  252
grey100 255  255  255
honeydew 240  255  240
honeydew1 240  255  240
honeydew2 224  238  224
<table>
<thead>
<tr>
<th>Color</th>
<th>R</th>
<th>G</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>honeydew3</td>
<td>193</td>
<td>205</td>
<td>193</td>
</tr>
<tr>
<td>honeydew4</td>
<td>131</td>
<td>139</td>
<td>131</td>
</tr>
<tr>
<td>hot pink</td>
<td>255</td>
<td>105</td>
<td>180</td>
</tr>
<tr>
<td>HotPink</td>
<td>255</td>
<td>105</td>
<td>180</td>
</tr>
<tr>
<td>HotPink1</td>
<td>255</td>
<td>110</td>
<td>180</td>
</tr>
<tr>
<td>HotPink2</td>
<td>238</td>
<td>106</td>
<td>167</td>
</tr>
<tr>
<td>HotPink3</td>
<td>205</td>
<td>96</td>
<td>144</td>
</tr>
<tr>
<td>HotPink4</td>
<td>139</td>
<td>58</td>
<td>98</td>
</tr>
<tr>
<td>indian red</td>
<td>205</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>IndianRed</td>
<td>205</td>
<td>92</td>
<td>92</td>
</tr>
<tr>
<td>IndianRed1</td>
<td>255</td>
<td>106</td>
<td>106</td>
</tr>
<tr>
<td>IndianRed2</td>
<td>238</td>
<td>99</td>
<td>99</td>
</tr>
<tr>
<td>IndianRed3</td>
<td>205</td>
<td>85</td>
<td>85</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>----------------------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>IndianRed4</td>
<td>139</td>
<td>58</td>
<td>58</td>
</tr>
<tr>
<td>ivory</td>
<td>255</td>
<td>255</td>
<td>240</td>
</tr>
<tr>
<td>ivory1</td>
<td>255</td>
<td>255</td>
<td>240</td>
</tr>
<tr>
<td>ivory2</td>
<td>238</td>
<td>238</td>
<td>224</td>
</tr>
<tr>
<td>ivory3</td>
<td>205</td>
<td>205</td>
<td>193</td>
</tr>
<tr>
<td>ivory4</td>
<td>139</td>
<td>139</td>
<td>131</td>
</tr>
<tr>
<td>khaki</td>
<td>240</td>
<td>230</td>
<td>140</td>
</tr>
<tr>
<td>khaki1</td>
<td>255</td>
<td>246</td>
<td>143</td>
</tr>
<tr>
<td>khaki2</td>
<td>238</td>
<td>230</td>
<td>133</td>
</tr>
<tr>
<td>khaki3</td>
<td>205</td>
<td>198</td>
<td>115</td>
</tr>
<tr>
<td>khaki4</td>
<td>139</td>
<td>134</td>
<td>78</td>
</tr>
<tr>
<td>lavender</td>
<td>230</td>
<td>230</td>
<td>250</td>
</tr>
<tr>
<td>lavender blush</td>
<td>255</td>
<td>240</td>
<td>245</td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>---------------------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>LavenderBlush</td>
<td>255</td>
<td>240</td>
<td>245</td>
</tr>
<tr>
<td>LavenderBlush1</td>
<td>255</td>
<td>240</td>
<td>245</td>
</tr>
<tr>
<td>LavenderBlush2</td>
<td>238</td>
<td>224</td>
<td>229</td>
</tr>
<tr>
<td>LavenderBlush3</td>
<td>205</td>
<td>193</td>
<td>197</td>
</tr>
<tr>
<td>LavenderBlush4</td>
<td>139</td>
<td>131</td>
<td>134</td>
</tr>
<tr>
<td>lawn green</td>
<td>124</td>
<td>252</td>
<td>0</td>
</tr>
<tr>
<td>LawnGreen</td>
<td>124</td>
<td>252</td>
<td>0</td>
</tr>
<tr>
<td>lemon chiffon</td>
<td>255</td>
<td>250</td>
<td>205</td>
</tr>
<tr>
<td>LemonChiffon</td>
<td>255</td>
<td>250</td>
<td>205</td>
</tr>
<tr>
<td>LemonChiffon1</td>
<td>255</td>
<td>250</td>
<td>205</td>
</tr>
<tr>
<td>LemonChiffon2</td>
<td>238</td>
<td>233</td>
<td>191</td>
</tr>
<tr>
<td>LemonChiffon3</td>
<td>205</td>
<td>201</td>
<td>165</td>
</tr>
<tr>
<td>LemonChiffon4</td>
<td>139</td>
<td>137</td>
<td>112</td>
</tr>
<tr>
<td>Color Description</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>light blue</td>
<td>173</td>
<td>216</td>
<td>230</td>
</tr>
<tr>
<td>light coral</td>
<td>240</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>light cyan</td>
<td>224</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>light goldenrod</td>
<td>238</td>
<td>221</td>
<td>130</td>
</tr>
<tr>
<td>light goldenrod yellow</td>
<td>250</td>
<td>250</td>
<td>210</td>
</tr>
<tr>
<td>light gray</td>
<td>211</td>
<td>211</td>
<td>211</td>
</tr>
<tr>
<td>light green</td>
<td>144</td>
<td>238</td>
<td>144</td>
</tr>
<tr>
<td>light grey</td>
<td>211</td>
<td>211</td>
<td>211</td>
</tr>
<tr>
<td>light pink</td>
<td>255</td>
<td>182</td>
<td>193</td>
</tr>
<tr>
<td>light salmon</td>
<td>255</td>
<td>160</td>
<td>122</td>
</tr>
<tr>
<td>light sea green</td>
<td>32</td>
<td>178</td>
<td>170</td>
</tr>
<tr>
<td>light sky blue</td>
<td>135</td>
<td>206</td>
<td>250</td>
</tr>
<tr>
<td>light slate blue</td>
<td>132</td>
<td>112</td>
<td>255</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>light slate gray</td>
<td>119</td>
<td>136</td>
<td>153</td>
</tr>
<tr>
<td>light slate grey</td>
<td>119</td>
<td>136</td>
<td>153</td>
</tr>
<tr>
<td>light steel blue</td>
<td>176</td>
<td>196</td>
<td>222</td>
</tr>
<tr>
<td>light yellow</td>
<td>255</td>
<td>255</td>
<td>224</td>
</tr>
<tr>
<td>LightBlue</td>
<td>173</td>
<td>216</td>
<td>230</td>
</tr>
<tr>
<td>LightBlue1</td>
<td>191</td>
<td>239</td>
<td>255</td>
</tr>
<tr>
<td>LightBlue2</td>
<td>178</td>
<td>223</td>
<td>238</td>
</tr>
<tr>
<td>LightBlue3</td>
<td>154</td>
<td>192</td>
<td>205</td>
</tr>
<tr>
<td>LightBlue4</td>
<td>104</td>
<td>131</td>
<td>139</td>
</tr>
<tr>
<td>LightCoral</td>
<td>240</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>LightCyan</td>
<td>224</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>LightCyan1</td>
<td>224</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>LightCyan2</td>
<td>209</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>LightCyan3</td>
<td>180</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>LightCyan4</td>
<td>122</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>LightGoldenrod</td>
<td>238</td>
<td>221</td>
<td>130</td>
</tr>
<tr>
<td>LightGoldenrod1</td>
<td>255</td>
<td>236</td>
<td>139</td>
</tr>
<tr>
<td>LightGoldenrod2</td>
<td>238</td>
<td>220</td>
<td>130</td>
</tr>
<tr>
<td>LightGoldenrod3</td>
<td>205</td>
<td>190</td>
<td>112</td>
</tr>
<tr>
<td>LightGoldenrod4</td>
<td>139</td>
<td>129</td>
<td>76</td>
</tr>
<tr>
<td>LightGoldenrodYellow</td>
<td>250</td>
<td>250</td>
<td>210</td>
</tr>
<tr>
<td>LightGray</td>
<td>211</td>
<td>211</td>
<td>211</td>
</tr>
<tr>
<td>LightGreen</td>
<td>144</td>
<td>238</td>
<td>144</td>
</tr>
<tr>
<td>LightGrey</td>
<td>211</td>
<td>211</td>
<td>211</td>
</tr>
<tr>
<td>LightPink</td>
<td>255</td>
<td>182</td>
<td>193</td>
</tr>
<tr>
<td>LightPink1</td>
<td>255</td>
<td>174</td>
<td>185</td>
</tr>
<tr>
<td>Color Name</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>LightPink2</td>
<td>238</td>
<td>162</td>
<td>173</td>
</tr>
<tr>
<td>LightPink3</td>
<td>205</td>
<td>140</td>
<td>149</td>
</tr>
<tr>
<td>LightPink4</td>
<td>139</td>
<td>95</td>
<td>101</td>
</tr>
<tr>
<td>LightSalmon</td>
<td>255</td>
<td>160</td>
<td>122</td>
</tr>
<tr>
<td>LightSalmon1</td>
<td>255</td>
<td>160</td>
<td>122</td>
</tr>
<tr>
<td>LightSalmon2</td>
<td>238</td>
<td>149</td>
<td>114</td>
</tr>
<tr>
<td>LightSalmon3</td>
<td>205</td>
<td>129</td>
<td>98</td>
</tr>
<tr>
<td>LightSalmon4</td>
<td>139</td>
<td>87</td>
<td>66</td>
</tr>
<tr>
<td>LightSeaGreen</td>
<td>32</td>
<td>178</td>
<td>170</td>
</tr>
<tr>
<td>LightSkyBlue</td>
<td>135</td>
<td>206</td>
<td>250</td>
</tr>
<tr>
<td>LightSkyBlue1</td>
<td>176</td>
<td>226</td>
<td>255</td>
</tr>
<tr>
<td>LightSkyBlue2</td>
<td>164</td>
<td>211</td>
<td>238</td>
</tr>
<tr>
<td>LightSkyBlue3</td>
<td>141</td>
<td>182</td>
<td>205</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>LightSkyBlue4</td>
<td>96</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td>LightSlateBlue</td>
<td>132</td>
<td>112</td>
<td>255</td>
</tr>
<tr>
<td>LightSlateGray</td>
<td>119</td>
<td>136</td>
<td>153</td>
</tr>
<tr>
<td>LightSlateGrey</td>
<td>119</td>
<td>136</td>
<td>153</td>
</tr>
<tr>
<td>LightSteelBlue</td>
<td>176</td>
<td>196</td>
<td>222</td>
</tr>
<tr>
<td>LightSteelBlue1</td>
<td>202</td>
<td>225</td>
<td>255</td>
</tr>
<tr>
<td>LightSteelBlue2</td>
<td>188</td>
<td>210</td>
<td>238</td>
</tr>
<tr>
<td>LightSteelBlue3</td>
<td>162</td>
<td>181</td>
<td>205</td>
</tr>
<tr>
<td>LightSteelBlue4</td>
<td>110</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td>LightYellow</td>
<td>255</td>
<td>255</td>
<td>224</td>
</tr>
<tr>
<td>LightYellow1</td>
<td>255</td>
<td>255</td>
<td>224</td>
</tr>
<tr>
<td>LightYellow2</td>
<td>238</td>
<td>238</td>
<td>209</td>
</tr>
<tr>
<td>LightYellow3</td>
<td>205</td>
<td>205</td>
<td>180</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>----------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>LightYellow4</td>
<td>139</td>
<td>139</td>
<td>122</td>
</tr>
<tr>
<td>lime green</td>
<td>50</td>
<td>205</td>
<td>50</td>
</tr>
<tr>
<td>LimeGreen</td>
<td>50</td>
<td>205</td>
<td>50</td>
</tr>
<tr>
<td>linen</td>
<td>250</td>
<td>240</td>
<td>230</td>
</tr>
<tr>
<td>magenta</td>
<td>255</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>magenta1</td>
<td>255</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>magenta2</td>
<td>238</td>
<td>0</td>
<td>238</td>
</tr>
<tr>
<td>magenta3</td>
<td>205</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>magenta4</td>
<td>139</td>
<td>0</td>
<td>139</td>
</tr>
<tr>
<td>maroon</td>
<td>176</td>
<td>48</td>
<td>96</td>
</tr>
<tr>
<td>maroon1</td>
<td>255</td>
<td>52</td>
<td>179</td>
</tr>
<tr>
<td>maroon2</td>
<td>238</td>
<td>48</td>
<td>167</td>
</tr>
<tr>
<td>maroon3</td>
<td>205</td>
<td>41</td>
<td>144</td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>maroon4</td>
<td>139</td>
<td>28</td>
<td>98</td>
</tr>
<tr>
<td>medium aquamarine</td>
<td>102</td>
<td>205</td>
<td>170</td>
</tr>
<tr>
<td>medium blue</td>
<td>0</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>medium orchid</td>
<td>186</td>
<td>85</td>
<td>211</td>
</tr>
<tr>
<td>medium purple</td>
<td>147</td>
<td>112</td>
<td>219</td>
</tr>
<tr>
<td>medium sea green</td>
<td>60</td>
<td>179</td>
<td>113</td>
</tr>
<tr>
<td>medium slate blue</td>
<td>123</td>
<td>104</td>
<td>238</td>
</tr>
<tr>
<td>medium spring green</td>
<td>0</td>
<td>250</td>
<td>154</td>
</tr>
<tr>
<td>medium turquoise</td>
<td>72</td>
<td>209</td>
<td>204</td>
</tr>
<tr>
<td>medium violet red</td>
<td>199</td>
<td>21</td>
<td>133</td>
</tr>
<tr>
<td>MediumAquamarine</td>
<td>102</td>
<td>205</td>
<td>170</td>
</tr>
<tr>
<td>MediumBlue</td>
<td>0</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>MediumOrchid</td>
<td>186</td>
<td>85</td>
<td>211</td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>MediumOrchid1</td>
<td>224</td>
<td>102</td>
<td>255</td>
</tr>
<tr>
<td>MediumOrchid2</td>
<td>209</td>
<td>95</td>
<td>238</td>
</tr>
<tr>
<td>MediumOrchid3</td>
<td>180</td>
<td>82</td>
<td>205</td>
</tr>
<tr>
<td>MediumOrchid4</td>
<td>122</td>
<td>55</td>
<td>139</td>
</tr>
<tr>
<td>MediumPurple</td>
<td>147</td>
<td>112</td>
<td>219</td>
</tr>
<tr>
<td>MediumPurple1</td>
<td>171</td>
<td>130</td>
<td>255</td>
</tr>
<tr>
<td>MediumPurple2</td>
<td>159</td>
<td>121</td>
<td>238</td>
</tr>
<tr>
<td>MediumPurple3</td>
<td>137</td>
<td>104</td>
<td>205</td>
</tr>
<tr>
<td>MediumPurple4</td>
<td>93</td>
<td>71</td>
<td>139</td>
</tr>
<tr>
<td>MediumSeaGreen</td>
<td>60</td>
<td>179</td>
<td>113</td>
</tr>
<tr>
<td>MediumSlateBlue</td>
<td>123</td>
<td>104</td>
<td>238</td>
</tr>
<tr>
<td>MediumSpringGreen</td>
<td>0</td>
<td>250</td>
<td>154</td>
</tr>
<tr>
<td>MediumTurquoise</td>
<td>72</td>
<td>209</td>
<td>204</td>
</tr>
<tr>
<td>Color</td>
<td>RGB Values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MediumVioletRed</td>
<td>199, 21, 133</td>
<td></td>
<td></td>
</tr>
<tr>
<td>midnight blue</td>
<td>25, 25, 112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MidnightBlue</td>
<td>25, 25, 112</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mint cream</td>
<td>245, 255, 250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MintCream</td>
<td>245, 255, 250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>misty rose</td>
<td>255, 228, 225</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose</td>
<td>255, 228, 225</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose1</td>
<td>255, 228, 225</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose2</td>
<td>238, 213, 210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose3</td>
<td>205, 183, 181</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MistyRose4</td>
<td>139, 125, 123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>moccasin</td>
<td>255, 228, 181</td>
<td></td>
<td></td>
</tr>
<tr>
<td>navajo white</td>
<td>255, 222, 173</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>------------------</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>NavajoWhite</td>
<td>255</td>
<td>222</td>
<td>173</td>
</tr>
<tr>
<td>NavajoWhite1</td>
<td>255</td>
<td>222</td>
<td>173</td>
</tr>
<tr>
<td>NavajoWhite2</td>
<td>238</td>
<td>207</td>
<td>161</td>
</tr>
<tr>
<td>NavajoWhite3</td>
<td>205</td>
<td>179</td>
<td>139</td>
</tr>
<tr>
<td>NavajoWhite4</td>
<td>139</td>
<td>121</td>
<td>94</td>
</tr>
<tr>
<td>navy</td>
<td>0</td>
<td>0</td>
<td>128</td>
</tr>
<tr>
<td>navy blue</td>
<td>0</td>
<td>0</td>
<td>128</td>
</tr>
<tr>
<td>NavyBlue</td>
<td>0</td>
<td>0</td>
<td>128</td>
</tr>
<tr>
<td>old lace</td>
<td>253</td>
<td>245</td>
<td>230</td>
</tr>
<tr>
<td>OldLace</td>
<td>253</td>
<td>245</td>
<td>230</td>
</tr>
<tr>
<td>olive drab</td>
<td>107</td>
<td>142</td>
<td>35</td>
</tr>
<tr>
<td>OliveDrab</td>
<td>107</td>
<td>142</td>
<td>35</td>
</tr>
<tr>
<td>OliveDrab1</td>
<td>192</td>
<td>255</td>
<td>62</td>
</tr>
<tr>
<td>Color Name</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>---------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>OliveDrab2</td>
<td>179</td>
<td>238</td>
<td>58</td>
</tr>
<tr>
<td>OliveDrab3</td>
<td>154</td>
<td>205</td>
<td>50</td>
</tr>
<tr>
<td>OliveDrab4</td>
<td>105</td>
<td>139</td>
<td>34</td>
</tr>
<tr>
<td>orange</td>
<td>255</td>
<td>165</td>
<td>0</td>
</tr>
<tr>
<td>orange red</td>
<td>255</td>
<td>69</td>
<td>0</td>
</tr>
<tr>
<td>orange1</td>
<td>255</td>
<td>165</td>
<td>0</td>
</tr>
<tr>
<td>orange2</td>
<td>238</td>
<td>154</td>
<td>0</td>
</tr>
<tr>
<td>orange3</td>
<td>205</td>
<td>133</td>
<td>0</td>
</tr>
<tr>
<td>orange4</td>
<td>139</td>
<td>90</td>
<td>0</td>
</tr>
<tr>
<td>OrangeRed</td>
<td>255</td>
<td>69</td>
<td>0</td>
</tr>
<tr>
<td>OrangeRed1</td>
<td>255</td>
<td>69</td>
<td>0</td>
</tr>
<tr>
<td>OrangeRed2</td>
<td>238</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>OrangeRed3</td>
<td>205</td>
<td>55</td>
<td>0</td>
</tr>
<tr>
<td>Color Name</td>
<td>RGB Values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OrangeRed4</td>
<td>139 37 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid</td>
<td>218 112 214</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid1</td>
<td>255 131 250</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid2</td>
<td>238 122 233</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid3</td>
<td>205 105 201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>orchid4</td>
<td>139 71 137</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pale goldenrod</td>
<td>238 232 170</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pale green</td>
<td>152 251 152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pale turquoise</td>
<td>175 238 238</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pale violet red</td>
<td>219 112 147</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaleGoldenrod</td>
<td>238 232 170</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaleGreen</td>
<td>152 251 152</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PaleGreen1</td>
<td>154 255 154</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>PaleGreen2</td>
<td>144</td>
<td>238</td>
<td>144</td>
</tr>
<tr>
<td>PaleGreen3</td>
<td>124</td>
<td>205</td>
<td>124</td>
</tr>
<tr>
<td>PaleGreen4</td>
<td>84</td>
<td>139</td>
<td>84</td>
</tr>
<tr>
<td>PaleTurquoise</td>
<td>175</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>PaleTurquoise1</td>
<td>187</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>PaleTurquoise2</td>
<td>174</td>
<td>238</td>
<td>238</td>
</tr>
<tr>
<td>PaleTurquoise3</td>
<td>150</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>PaleTurquoise4</td>
<td>102</td>
<td>139</td>
<td>139</td>
</tr>
<tr>
<td>PaleVioletRed</td>
<td>219</td>
<td>112</td>
<td>147</td>
</tr>
<tr>
<td>PaleVioletRed1</td>
<td>255</td>
<td>130</td>
<td>171</td>
</tr>
<tr>
<td>PaleVioletRed2</td>
<td>238</td>
<td>121</td>
<td>159</td>
</tr>
<tr>
<td>PaleVioletRed3</td>
<td>205</td>
<td>104</td>
<td>127</td>
</tr>
<tr>
<td>PaleVioletRed4</td>
<td>139</td>
<td>71</td>
<td>93</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>--------------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>papaya whip</td>
<td>255</td>
<td>239</td>
<td>213</td>
</tr>
<tr>
<td>PapayaWhip</td>
<td>255</td>
<td>239</td>
<td>213</td>
</tr>
<tr>
<td>peach puff</td>
<td>255</td>
<td>218</td>
<td>185</td>
</tr>
<tr>
<td>PeachPuff</td>
<td>255</td>
<td>218</td>
<td>185</td>
</tr>
<tr>
<td>PeachPuff1</td>
<td>255</td>
<td>218</td>
<td>185</td>
</tr>
<tr>
<td>PeachPuff2</td>
<td>238</td>
<td>203</td>
<td>173</td>
</tr>
<tr>
<td>PeachPuff3</td>
<td>205</td>
<td>175</td>
<td>149</td>
</tr>
<tr>
<td>PeachPuff4</td>
<td>139</td>
<td>119</td>
<td>101</td>
</tr>
<tr>
<td>peru</td>
<td>205</td>
<td>133</td>
<td>63</td>
</tr>
<tr>
<td>pink</td>
<td>255</td>
<td>192</td>
<td>203</td>
</tr>
<tr>
<td>pink1</td>
<td>255</td>
<td>181</td>
<td>197</td>
</tr>
<tr>
<td>pink2</td>
<td>238</td>
<td>169</td>
<td>184</td>
</tr>
<tr>
<td>pink3</td>
<td>205</td>
<td>145</td>
<td>158</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>----------------</td>
<td>-----</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>pink4</td>
<td>139</td>
<td>99</td>
<td>108</td>
</tr>
<tr>
<td>plum</td>
<td>221</td>
<td>160</td>
<td>221</td>
</tr>
<tr>
<td>plum1</td>
<td>255</td>
<td>187</td>
<td>255</td>
</tr>
<tr>
<td>plum2</td>
<td>238</td>
<td>174</td>
<td>238</td>
</tr>
<tr>
<td>plum3</td>
<td>205</td>
<td>150</td>
<td>205</td>
</tr>
<tr>
<td>plum4</td>
<td>139</td>
<td>102</td>
<td>139</td>
</tr>
<tr>
<td>powder blue</td>
<td>176</td>
<td>224</td>
<td>230</td>
</tr>
<tr>
<td>PowderBlue</td>
<td>176</td>
<td>224</td>
<td>230</td>
</tr>
<tr>
<td>purple</td>
<td>160</td>
<td>32</td>
<td>240</td>
</tr>
<tr>
<td>purple1</td>
<td>155</td>
<td>48</td>
<td>255</td>
</tr>
<tr>
<td>purple2</td>
<td>145</td>
<td>44</td>
<td>238</td>
</tr>
<tr>
<td>purple3</td>
<td>125</td>
<td>38</td>
<td>205</td>
</tr>
<tr>
<td>purple4</td>
<td>85</td>
<td>26</td>
<td>139</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>red</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>red1</td>
<td>255</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>red2</td>
<td>238</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>red3</td>
<td>205</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>red4</td>
<td>139</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rosy brown</td>
<td>188</td>
<td>143</td>
<td>143</td>
</tr>
<tr>
<td>RosyBrown</td>
<td>188</td>
<td>143</td>
<td>143</td>
</tr>
<tr>
<td>RosyBrown1</td>
<td>255</td>
<td>193</td>
<td>193</td>
</tr>
<tr>
<td>RosyBrown2</td>
<td>238</td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>RosyBrown3</td>
<td>205</td>
<td>155</td>
<td>155</td>
</tr>
<tr>
<td>RosyBrown4</td>
<td>139</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>royal blue</td>
<td>65</td>
<td>105</td>
<td>225</td>
</tr>
<tr>
<td>RoyalBlue</td>
<td>65</td>
<td>105</td>
<td>225</td>
</tr>
<tr>
<td>Color Name</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>RoyalBlue1</td>
<td>72</td>
<td>118</td>
<td>255</td>
</tr>
<tr>
<td>RoyalBlue2</td>
<td>67</td>
<td>110</td>
<td>238</td>
</tr>
<tr>
<td>RoyalBlue3</td>
<td>58</td>
<td>95</td>
<td>205</td>
</tr>
<tr>
<td>RoyalBlue4</td>
<td>39</td>
<td>64</td>
<td>139</td>
</tr>
<tr>
<td>saddle brown</td>
<td>139</td>
<td>69</td>
<td>19</td>
</tr>
<tr>
<td>SaddleBrown</td>
<td>139</td>
<td>69</td>
<td>19</td>
</tr>
<tr>
<td>salmon</td>
<td>250</td>
<td>128</td>
<td>114</td>
</tr>
<tr>
<td>salmon1</td>
<td>255</td>
<td>140</td>
<td>105</td>
</tr>
<tr>
<td>salmon2</td>
<td>238</td>
<td>130</td>
<td>98</td>
</tr>
<tr>
<td>salmon3</td>
<td>205</td>
<td>112</td>
<td>84</td>
</tr>
<tr>
<td>salmon4</td>
<td>139</td>
<td>76</td>
<td>57</td>
</tr>
<tr>
<td>sandy brown</td>
<td>244</td>
<td>164</td>
<td>96</td>
</tr>
<tr>
<td>SandyBrown</td>
<td>244</td>
<td>164</td>
<td>96</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>---------</td>
<td>-----</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>sea green</td>
<td>46</td>
<td>139</td>
<td>87</td>
</tr>
<tr>
<td>SeaGreen</td>
<td>46</td>
<td>139</td>
<td>87</td>
</tr>
<tr>
<td>SeaGreen1</td>
<td>84</td>
<td>255</td>
<td>159</td>
</tr>
<tr>
<td>SeaGreen2</td>
<td>78</td>
<td>238</td>
<td>148</td>
</tr>
<tr>
<td>SeaGreen3</td>
<td>67</td>
<td>205</td>
<td>128</td>
</tr>
<tr>
<td>SeaGreen4</td>
<td>46</td>
<td>139</td>
<td>87</td>
</tr>
<tr>
<td>seashell</td>
<td>255</td>
<td>245</td>
<td>238</td>
</tr>
<tr>
<td>seashell1</td>
<td>255</td>
<td>245</td>
<td>238</td>
</tr>
<tr>
<td>seashell2</td>
<td>238</td>
<td>229</td>
<td>222</td>
</tr>
<tr>
<td>seashell3</td>
<td>205</td>
<td>197</td>
<td>191</td>
</tr>
<tr>
<td>seashell4</td>
<td>139</td>
<td>134</td>
<td>130</td>
</tr>
<tr>
<td>sienna</td>
<td>160</td>
<td>82</td>
<td>45</td>
</tr>
<tr>
<td>sienna1</td>
<td>255</td>
<td>130</td>
<td>71</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>sienna2</td>
<td>238</td>
<td>121</td>
<td>66</td>
</tr>
<tr>
<td>sienna3</td>
<td>205</td>
<td>104</td>
<td>57</td>
</tr>
<tr>
<td>sienna4</td>
<td>139</td>
<td>71</td>
<td>38</td>
</tr>
<tr>
<td>sky blue</td>
<td>135</td>
<td>206</td>
<td>235</td>
</tr>
<tr>
<td>SkyBlue</td>
<td>135</td>
<td>206</td>
<td>235</td>
</tr>
<tr>
<td>SkyBlue1</td>
<td>135</td>
<td>206</td>
<td>255</td>
</tr>
<tr>
<td>SkyBlue2</td>
<td>126</td>
<td>192</td>
<td>238</td>
</tr>
<tr>
<td>SkyBlue3</td>
<td>108</td>
<td>166</td>
<td>205</td>
</tr>
<tr>
<td>SkyBlue4</td>
<td>74</td>
<td>112</td>
<td>139</td>
</tr>
<tr>
<td>slate blue</td>
<td>106</td>
<td>90</td>
<td>205</td>
</tr>
<tr>
<td>slate gray</td>
<td>112</td>
<td>128</td>
<td>144</td>
</tr>
<tr>
<td>slate grey</td>
<td>112</td>
<td>128</td>
<td>144</td>
</tr>
<tr>
<td>SlateBlue</td>
<td>106</td>
<td>90</td>
<td>205</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>SlateBlue1</td>
<td>131</td>
<td>111</td>
<td>255</td>
</tr>
<tr>
<td>SlateBlue2</td>
<td>122</td>
<td>103</td>
<td>238</td>
</tr>
<tr>
<td>SlateBlue3</td>
<td>105</td>
<td>89</td>
<td>205</td>
</tr>
<tr>
<td>SlateBlue4</td>
<td>71</td>
<td>60</td>
<td>139</td>
</tr>
<tr>
<td>SlateGray</td>
<td>112</td>
<td>128</td>
<td>144</td>
</tr>
<tr>
<td>SlateGray1</td>
<td>198</td>
<td>226</td>
<td>255</td>
</tr>
<tr>
<td>SlateGray2</td>
<td>185</td>
<td>211</td>
<td>238</td>
</tr>
<tr>
<td>SlateGray3</td>
<td>159</td>
<td>182</td>
<td>205</td>
</tr>
<tr>
<td>SlateGray4</td>
<td>108</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td>SlateGrey</td>
<td>112</td>
<td>128</td>
<td>144</td>
</tr>
<tr>
<td>snow</td>
<td>255</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>snow1</td>
<td>255</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>snow2</td>
<td>238</td>
<td>233</td>
<td>233</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>---------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>snow3</td>
<td>205</td>
<td>201</td>
<td>201</td>
</tr>
<tr>
<td>snow4</td>
<td>139</td>
<td>137</td>
<td>137</td>
</tr>
<tr>
<td>spring green</td>
<td>0</td>
<td>255</td>
<td>127</td>
</tr>
<tr>
<td>SpringGreen</td>
<td>0</td>
<td>255</td>
<td>127</td>
</tr>
<tr>
<td>SpringGreen1</td>
<td>0</td>
<td>255</td>
<td>127</td>
</tr>
<tr>
<td>SpringGreen2</td>
<td>0</td>
<td>238</td>
<td>118</td>
</tr>
<tr>
<td>SpringGreen3</td>
<td>0</td>
<td>205</td>
<td>102</td>
</tr>
<tr>
<td>SpringGreen4</td>
<td>0</td>
<td>139</td>
<td>69</td>
</tr>
<tr>
<td>steel blue</td>
<td>70</td>
<td>130</td>
<td>180</td>
</tr>
<tr>
<td>SteelBlue</td>
<td>70</td>
<td>130</td>
<td>180</td>
</tr>
<tr>
<td>SteelBlue1</td>
<td>99</td>
<td>184</td>
<td>255</td>
</tr>
<tr>
<td>SteelBlue2</td>
<td>92</td>
<td>172</td>
<td>238</td>
</tr>
<tr>
<td>SteelBlue3</td>
<td>79</td>
<td>148</td>
<td>205</td>
</tr>
<tr>
<td>Color</td>
<td>H</td>
<td>S</td>
<td>L</td>
</tr>
<tr>
<td>------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>SteelBlue4</td>
<td>54</td>
<td>100</td>
<td>139</td>
</tr>
<tr>
<td>tan</td>
<td>210</td>
<td>180</td>
<td>140</td>
</tr>
<tr>
<td>tan1</td>
<td>255</td>
<td>165</td>
<td>79</td>
</tr>
<tr>
<td>tan2</td>
<td>238</td>
<td>154</td>
<td>73</td>
</tr>
<tr>
<td>tan3</td>
<td>205</td>
<td>133</td>
<td>63</td>
</tr>
<tr>
<td>tan4</td>
<td>139</td>
<td>90</td>
<td>43</td>
</tr>
<tr>
<td>thistle</td>
<td>216</td>
<td>191</td>
<td>216</td>
</tr>
<tr>
<td>thistle1</td>
<td>255</td>
<td>225</td>
<td>255</td>
</tr>
<tr>
<td>thistle2</td>
<td>238</td>
<td>210</td>
<td>238</td>
</tr>
<tr>
<td>thistle3</td>
<td>205</td>
<td>181</td>
<td>205</td>
</tr>
<tr>
<td>thistle4</td>
<td>139</td>
<td>123</td>
<td>139</td>
</tr>
<tr>
<td>tomato</td>
<td>255</td>
<td>99</td>
<td>71</td>
</tr>
<tr>
<td>tomato1</td>
<td>255</td>
<td>99</td>
<td>71</td>
</tr>
<tr>
<td>Color</td>
<td>R</td>
<td>G</td>
<td>B</td>
</tr>
<tr>
<td>---------------</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>tomato2</td>
<td>238</td>
<td>92</td>
<td>66</td>
</tr>
<tr>
<td>tomato3</td>
<td>205</td>
<td>79</td>
<td>57</td>
</tr>
<tr>
<td>tomato4</td>
<td>139</td>
<td>54</td>
<td>38</td>
</tr>
<tr>
<td>turquoise</td>
<td>64</td>
<td>224</td>
<td>208</td>
</tr>
<tr>
<td>turquoise1</td>
<td>0</td>
<td>245</td>
<td>255</td>
</tr>
<tr>
<td>turquoise2</td>
<td>0</td>
<td>229</td>
<td>238</td>
</tr>
<tr>
<td>turquoise3</td>
<td>0</td>
<td>197</td>
<td>205</td>
</tr>
<tr>
<td>turquoise4</td>
<td>0</td>
<td>134</td>
<td>139</td>
</tr>
<tr>
<td>violet</td>
<td>238</td>
<td>130</td>
<td>238</td>
</tr>
<tr>
<td>violet red</td>
<td>208</td>
<td>32</td>
<td>144</td>
</tr>
<tr>
<td>VioletRed</td>
<td>208</td>
<td>32</td>
<td>144</td>
</tr>
<tr>
<td>VioletRed1</td>
<td>255</td>
<td>62</td>
<td>150</td>
</tr>
<tr>
<td>VioletRed2</td>
<td>238</td>
<td>58</td>
<td>140</td>
</tr>
<tr>
<td>Color</td>
<td>Red</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>---------------</td>
<td>------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>VioletRed3</td>
<td>205</td>
<td>50</td>
<td>120</td>
</tr>
<tr>
<td>VioletRed4</td>
<td>139</td>
<td>34</td>
<td>82</td>
</tr>
<tr>
<td>wheat</td>
<td>245</td>
<td>222</td>
<td>179</td>
</tr>
<tr>
<td>wheat1</td>
<td>255</td>
<td>231</td>
<td>186</td>
</tr>
<tr>
<td>wheat2</td>
<td>238</td>
<td>216</td>
<td>174</td>
</tr>
<tr>
<td>wheat3</td>
<td>205</td>
<td>186</td>
<td>150</td>
</tr>
<tr>
<td>wheat4</td>
<td>139</td>
<td>126</td>
<td>102</td>
</tr>
<tr>
<td>white</td>
<td>255</td>
<td>255</td>
<td>255</td>
</tr>
<tr>
<td>white smoke</td>
<td>245</td>
<td>245</td>
<td>245</td>
</tr>
<tr>
<td>WhiteSmoke</td>
<td>245</td>
<td>245</td>
<td>245</td>
</tr>
<tr>
<td>yellow</td>
<td>255</td>
<td>255</td>
<td>0</td>
</tr>
<tr>
<td>yellow green</td>
<td>154</td>
<td>205</td>
<td>50</td>
</tr>
<tr>
<td>yellow1</td>
<td>255</td>
<td>255</td>
<td>0</td>
</tr>
</tbody>
</table>
PORTABILITY ISSUES

Mac OS X
On Mac OS X, the following additional system colors are available (note that the actual color values depend on the currently active OS theme, and typically many of these will in fact be patterns rather than pure colors):

  systemActiveAreaFill

  systemAlertActiveText

  systemAlertBackgroundActive

  systemAlertBackgroundInactive

  systemAlertInactiveText

  systemAlternatePrimaryHighlightColor

  systemAppleGuideCoachmark
systemBevelActiveDark

systemBevelActiveLight

systemBevelButtonActiveText

systemBevelButtonInactiveText

systemBevelButtonPressedText

systemBevelButtonStickyActiveText

systemBevelButtonStickyInactiveText

systemBevelInactiveDark

systemBevelInactiveLight

systemBlack

systemBlackText

systemButtonActiveDarkHighlight

systemButtonActiveDarkShadow
systemButtonActiveLightHighlight

systemButtonActiveLightShadow

systemButtonFace

systemButtonFaceActive

systemButtonFaceInactive

systemButtonFacePressed

systemButtonFrame

systemButtonFrameActive

systemButtonFrameInactive

systemButtonInactiveDarkHighlight

systemButtonInactiveDarkShadow

systemButtonInactiveLightHighlight

systemButtonInactiveLightShadow
systemButtonPressedDarkHighlight

dystemButtonPressedDarkShadow

dystemButtonPressedLightHighlight

dystemButtonPressedLightShadow

dystemButtonText

dystemChasingArrows

dystemDialogActiveText

dystemDialogBackgroundActive

dystemDialogBackgroundInactive

dystemDialogInactiveText

dystemDocumentWindowBackground

dystemDocumentWindowTitleActiveText

dystemDocumentWindowTitleInactiveText
systemDragHilite

systemDrawerBackground

systemFinderWindowBackground

systemFocusHighlight

systemHighlight

systemHighlightAlternate

systemHighlightSecondary

systemHighlightText

systemIconLabelBackground

systemIconLabelBackgroundSelected

systemIconLabelSelectedText

systemIconLabelText

systemListViewBackground
systemListViewColumnDivider

systemListViewEvenRowBackground

systemListViewOddRowBackground

systemListViewSeparator

systemListViewSortColumnBackground

systemListViewText

systemListViewWindowHeaderBackground

systemMenu

systemMenuActive

systemMenuActiveText

systemMenuBackground

systemMenuBackgroundSelected

systemMenuDisabled
systemMenuItemActiveText

systemMenuItemDisabledText

systemMenuItemSelectedText

systemMenuText

systemMetalBackground

systemModelessDialogActiveText

systemModelessDialogBackgroundActive

systemModelessDialogBackgroundInactive

systemModelessDialogInactiveText

systemMovableModalBackground

systemMovableModalWindowTitleActiveText

systemMovableModalWindowTitleInactiveText

systemNotificationText
systemPopupWindowTitleActiveText

systemPopupWindowTitleInactiveText

systemPrimaryHighlightColor

systemPushButtonActiveText

systemPushButtonInactiveText

systemPushButtonPressedText

systemRootMenuActiveText

systemRootMenuDisabledText

systemRootMenuSelectedText

systemScrollBarDelimiterActive

systemScrollBarDelimiterInactive

systemSecondaryGroupBoxBackground

systemSecondaryHighlightColor
systemSheetBackground

systemSheetBackgroundOpaque

systemSheetBackgroundTransparent

systemStaticAreaFill

systemSystemDetailText

systemTabFrontActiveText

systemTabFrontInactiveText

systemTabNonFrontActiveText

systemTabNonFrontInactiveText

systemTabNonFrontPressedText

systemTabPaneBackground

systemToolbarBackground

systemTransparent
Windows

On Windows, the following additional system colors are available (note that the actual color values depend on the currently active OS theme):

3dDarkShadow  Highlight

3dLight  HighlightText
<table>
<thead>
<tr>
<th>ActiveBorder</th>
<th>InactiveBorder</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveCaption</td>
<td>InactiveCaption</td>
</tr>
<tr>
<td>AppWorkspace</td>
<td>InactiveCaptionText</td>
</tr>
<tr>
<td>Background</td>
<td>InfoBackground</td>
</tr>
<tr>
<td>ButtonFace</td>
<td>InfoText</td>
</tr>
<tr>
<td>ButtonHighlight</td>
<td>Menu</td>
</tr>
<tr>
<td>ButtonShadow</td>
<td>MenuText</td>
</tr>
<tr>
<td>ButtonText</td>
<td>Scrollbar</td>
</tr>
<tr>
<td>CaptionText</td>
<td>Window</td>
</tr>
<tr>
<td>DisabledText</td>
<td>WindowFrame</td>
</tr>
<tr>
<td>GrayText</td>
<td>WindowText</td>
</tr>
</tbody>
</table>

**SEE ALSO**

options, Tk_GetColor

**KEYWORDS**
color, option
NAME
listbox - Create and manipulate listbox widgets

SYNOPSIS

STANDARD OPTIONS
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-exportselection, exportSelection, ExportSelection
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-relief, relief, Relief
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-setgrid, setGrid, SetGrid
-takefocus, takeFocus, TakeFocus
-xscrollcommand, xScrollCommand, ScrollCommand
-yscrollcommand, yScrollCommand, ScrollCommand

WIDGET-SPECIFIC OPTIONS
-activestyle, activeStyle, ActiveStyle
-height, height, Height
-listvariable, listVariable, Variable
-selectmode, selectMode, SelectMode
-state, state, State
-width, width, Width

DESCRIPTION
INDICES
  number
  active
  anchor
  end
  @x,y

WIDGET COMMAND
  pathName activate index
  pathName bbox index
  pathName cget option
  pathName configure ?option? ?value option value ...?
  pathName curselection
  pathName delete first ?last?
  pathName get first ?last?
  pathName index index
  pathName insert index ?element element ...?
  pathName itemcget index option
  pathName itemconfigure index ?option? ?value? ?option value ...
  -background color
  -foreground color
  -selectbackground color
  -selectforeground color
  pathName nearest y
  pathName scan option args
    pathName scan mark x y
    pathName scan dragto x y.
  pathName see index
  pathName selection option arg
    pathName selection anchor index
    pathName selection clear first ?last?
    pathName selection includes index
    pathName selection set first ?last?
  pathName size
  pathName xview args
    pathName xview
    pathName xview index
`pathName xview moveto fraction`  
`pathName xview scroll number what`  
`pathName yview ?args?`  
`pathName yview`  
`pathName yview index`  
`pathName yview moveto fraction`  
`pathName yview scroll number what`  

**DEFAULT BINDINGS**

**SEE ALSO**

**KEYWORDS**

**NAME**

listbox - Create and manipulate listbox widgets

**SYNOPSIS**

`listbox pathName ?options?`

**STANDARD OPTIONS**

-`-background or -bg, background, Background`
-`-borderwidth or -bd, borderWidth, BorderWidth`
-`-cursor, cursor, Cursor`
-`-disabledforeground, disabledForeground, DisabledForeground`
-`-exportselection, exportSelection, ExportSelection`
-`-font, font, Font`
-`-foreground or -fg, foreground, Foreground`
-`-highlightbackground, highlightBackground, HighlightBackground`
-`-highlightcolor, highlightColor, HighlightColor`
-`-highlightthickness, highlightThickness, HighlightThickness`
-`-relief, relief, Relief`
-`-selectbackground, selectBackground, Foreground`
-`-selectborderwidth, selectBorderWidth, BorderWidth`
-`-selectforeground, selectForeground, Background`
-`-setgrid, setGrid, SetGrid`
-`-takefocus, takeFocus, TakeFocus`
-`-xscrollcommand, xScrollCommand, ScrollCommand`
**-yscrollcommand, yScrollCommand, ScrollCommand**

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: **-activestyle**  
Database Name: **activeStyle**  
Database Class: **ActiveStyle**  
Specifies the style in which to draw the active element. This must be one of **dotbox** (show a focus ring around the active element), **none** (no special indication of active element) or **underline** (underline the active element). The default is **underline** on Windows, and **dotbox** elsewhere.

Command-Line Name: **-height**  
Database Name: **height**  
Database Class: **Height**  
Specifies the desired height for the window, in lines. If zero or less, then the desired height for the window is made just large enough to hold all the elements in the listbox.

Command-Line Name: **-listvariable**  
Database Name: **listVariable**  
Database Class: **Variable**  
Specifies the name of a variable. The value of the variable is a list to be displayed inside the widget; if the variable value changes then the widget will automatically update itself to reflect the new value. Attempts to assign a variable with an invalid list value to **-listvariable** will cause an error. Attempts to unset a variable in use as a **-listvariable** will fail but will not generate an error.

Command-Line Name: **-selectmode**  
Database Name: **selectMode**  
Database Class: **SelectMode**  
Specifies one of several styles for manipulating the selection. The value of the option may be arbitrary, but the default bindings expect it to be either **single**, **browse**, **multiple**, or **extended**; the default value is **browse**.
Command-Line Name: `-state`
Database Name: `state`
Database Class: `State`
   Specifies one of two states for the listbox: `normal` or `disabled`. If the listbox is disabled then items may not be inserted or deleted, items are drawn in the `-disabledforeground` color, and selection cannot be modified and is not shown (though selection information is retained).

Command-Line Name: `-width`
Database Name: `width`
Database Class: `Width`
   Specifies the desired width for the window in characters. If the font does not have a uniform width then the width of the character "0" is used in translating from character units to screen units. If zero or less, then the desired width for the window is made just large enough to hold all the elements in the listbox.

**DESCRIPTION**

The `listbox` command creates a new window (given by the `pathName` argument) and makes it into a listbox widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the listbox such as its colors, font, text, and relief. The `listbox` command returns its `pathName` argument. At the time this command is invoked, there must not exist a window named `pathName`, but `pathName`'s parent must exist.

A listbox is a widget that displays a list of strings, one per line. When first created, a new listbox has no elements. Elements may be added or deleted using widget commands described below. In addition, one or more elements may be selected as described below. If a listbox is exporting its selection (see `exportSelection` option), then it will observe the standard X11 protocols for handling the selection. Listbox selections are available as type `STRING`; the value of the selection will be the text of the selected elements, with newlines separating the elements.

It is not necessary for all the elements to be displayed in the listbox.
window at once; commands described below may be used to change the view in the window. Listboxes allow scrolling in both directions using the standard `xScrollCommand` and `yScrollCommand` options. They also support scanning, as described below.

## INDICES

Many of the widget commands for listboxes take one or more indices as arguments. An index specifies a particular element of the listbox, in any of the following ways:

**number**
- Specifies the element as a numerical index, where 0 corresponds to the first element in the listbox.

**active**
- Indicates the element that has the location cursor. This element will be displayed as specified by `-activestyle` when the listbox has the keyboard focus, and it is specified with the `activate` widget command.

**anchor**
- Indicates the anchor point for the selection, which is set with the `selection anchor` widget command.

**end**
- Indicates the end of the listbox. For most commands this refers to the last element in the listbox, but for a few commands such as `index` and `insert` it refers to the element just after the last one.

**@x,y**
- Indicates the element that covers the point in the listbox window specified by x and y (in pixel coordinates). If no element covers that point, then the closest element to that point is used.

In the widget command descriptions below, arguments named `index`, `first`, and `last` always contain text indices in one of the above forms.
The **listbox** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for listbox widgets:

**pathName activate index**
Sets the active element to the one indicated by *index*. If *index* is outside the range of elements in the listbox then the closest element is activated. The active element is drawn as specified by **activestyle** when the widget has the input focus, and its index may be retrieved with the index **active**.

**pathName bbox index**
Returns a list of four numbers describing the bounding box of the text in the element given by *index*. The first two elements of the list give the x and y coordinates of the upper-left corner of the screen area covered by the text (specified in pixels relative to the widget) and the last two elements give the width and height of the area, in pixels. If no part of the element given by *index* is visible on the screen, or if *index* refers to a non-existent element, then the result is an empty string; if the element is partially visible, the result gives the full area of the element, including any parts that are not visible.

**pathName cget option**
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **listbox** command.

**pathName configure ?option? ?value option value ...?**
Query or modify the configuration options of the widget. If no *option*
is specified, returns a list describing all of the available options for 
opathName (see Tk_ConfigureInfo for information on the format of 
this list). If option is specified with no value, then the command 
returns a list describing the one named option (this list will be 
identical to the corresponding sublist of the value returned if no 
option is specified). If one or more option-value pairs are specified, 
then the command modifies the given widget option(s) to have the 
given value(s); in this case the command returns an empty string. 
Option may have any of the values accepted by the listbox 
command.

opathName curselection

Returns a list containing the numerical indices of all of the elements 
in the listbox that are currently selected. If there are no elements 
selected in the listbox then an empty string is returned.

opathName delete first ?last?

Deletes one or more elements of the listbox. First and last are 
indices specifying the first and last elements in the range to delete. 
If last is not specified it defaults to first, i.e. a single element is 
deleted.

opathName get first ?last?

If last is omitted, returns the contents of the listbox element 
indicated by first, or an empty string if first refers to a non-existent 
element. If last is specified, the command returns a list whose 
elements are all of the listbox elements between first and last, 
inclusive. Both first and last may have any of the standard forms for 
indices.

opathName index index

Returns the integer index value that corresponds to index. If index 
is end the return value is a count of the number of elements in the 
listbox (not the index of the last element).

opathName insert index ?element element ...?

Inserts zero or more new elements in the list just before the 
element given by index. If index is specified as end then the new
elements are added to the end of the list. Returns an empty string.

*pathName itemcget index option*

Returns the current value of the item configuration option given by *option*. *Option* may have any of the values accepted by the `listbox itemconfigure` command.

*pathName itemconfigure index ?option? ?value? ?option value ...?*

Query or modify the configuration options of an item in the listbox. If no *option* is specified, returns a list describing all of the available options for the item (see `Tk_ConfigureInfo` for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option-value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. The following options are currently supported for items:

- **-background** *color*
  *Color* specifies the background color to use when displaying the item. It may have any of the forms accepted by `Tk_GetColor`.

- **-foreground** *color*
  *Color* specifies the foreground color to use when displaying the item. It may have any of the forms accepted by `Tk_GetColor`.

- **-selectbackground** *color*
  *color* specifies the background color to use when displaying the item while it is selected. It may have any of the forms accepted by `Tk_GetColor`.

- **-selectforeground** *color*
  *color* specifies the foreground color to use when displaying the item while it is selected. It may have any of the forms accepted by `Tk_GetColor`.
**pathName nearest y**
Given a y-coordinate within the listbox window, this command returns the index of the (visible) listbox element nearest to that y-coordinate.

**pathName scan option args**
This command is used to implement scanning on listboxes. It has two forms, depending on `option`:

**pathName scan mark x y**
Records x and y and the current view in the listbox window; used in conjunction with later `scan dragto` commands. Typically this command is associated with a mouse button press in the widget. It returns an empty string.

**pathName scan dragto x y.**
This command computes the difference between its x and y arguments and the x and y arguments to the last `scan mark` command for the widget. It then adjusts the view by 10 times the difference in coordinates. This command is typically associated with mouse motion events in the widget, to produce the effect of dragging the list at high speed through the window. The return value is an empty string.

**pathName see index**
Adjust the view in the listbox so that the element given by `index` is visible. If the element is already visible then the command has no effect; if the element is near one edge of the window then the listbox scrolls to bring the element into view at the edge; otherwise the listbox scrolls to center the element.

**pathName selection option arg**
This command is used to adjust the selection within a listbox. It has several forms, depending on `option`:

**pathName selection anchor index**
Sets the selection anchor to the element given by `index`. If `index` refers to a non-existent element, then the closest
element is used. The selection anchor is the end of the selection that is fixed while dragging out a selection with the mouse. The index anchor may be used to refer to the anchor element.

`pathName selection clear first ?last?`
If any of the elements between `first` and `last` (inclusive) are selected, they are deselected. The selection state is not changed for elements outside this range.

`pathName selection includes index`
Returns 1 if the element indicated by `index` is currently selected, 0 if it is not.

`pathName selection set first ?last?`
Selects all of the elements in the range between `first` and `last`, inclusive, without affecting the selection state of elements outside that range.

`pathName size`
Returns a decimal string indicating the total number of elements in the listbox.

`pathName xview args`
This command is used to query and change the horizontal position of the information in the widget's window. It can take any of the following forms:

`pathName xview`
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the listbox's text is off-screen to the left, the middle 40% is visible in the window, and 40% of the text is off-screen to the right. These are the same values passed to scrollbars via the `-xscrollcommand` option.
**pathName xview index**

Adjusts the view in the window so that the character position given by *index* is displayed at the left edge of the window. Character positions are defined by the width of the character 0.

**pathName xview moveto fraction**

Adjusts the view in the window so that *fraction* of the total width of the listbox text is off-screen to the left. *fraction* must be a fraction between 0 and 1.

**pathName xview scroll number what**

This command shifts the view in the window left or right according to *number* and *what*. *Number* must be an integer. *What* must be either *units* or *pages* or an abbreviation of one of these. If *what* is *units*, the view adjusts left or right by *number* character units (the width of the 0 character) on the display; if it is *pages* then the view adjusts by *number* screenfuls. If *number* is negative then characters farther to the left become visible; if it is positive then characters farther to the right become visible.

**pathName yview ?args?**

This command is used to query and change the vertical position of the text in the widget's window. It can take any of the following forms:

**pathName yview**

Returns a list containing two elements, both of which are real fractions between 0 and 1. The first element gives the position of the listbox element at the top of the window, relative to the listbox as a whole (0.5 means it is halfway through the listbox, for example). The second element gives the position of the listbox element just after the last one in the window, relative to the listbox as a whole. These are the same values passed to scrollbars via the **-yscrollcommand** option.

**pathName yview index**

Adjusts the view in the window so that the element given by
*index* is displayed at the top of the window.

**pathName yview moveto fraction**
Adjusts the view in the window so that the element given by *fraction* appears at the top of the window. *Fraction* is a fraction between 0 and 1; 0 indicates the first element in the listbox, 0.33 indicates the element one-third the way through the listbox, and so on.

**pathName yview scroll number what**
This command adjusts the view in the window up or down according to *number* and *what*. *Number* must be an integer. *What* must be either *units* or *pages*. If *what* is *units*, the view adjusts up or down by *number* lines; if it is *pages* then the view adjusts by *number* screenfuls. If *number* is negative then earlier elements become visible; if it is positive then later elements become visible.

### DEFAULT BINDINGS

Tk automatically creates class bindings for listboxes that give them Motif-like behavior. Much of the behavior of a listbox is determined by its *selectMode* option, which selects one of four ways of dealing with the selection.

If the selection mode is *single* or *browse*, at most one element can be selected in the listbox at once. In both modes, clicking button 1 on an element selects it and deselects any other selected item. In *browse* mode it is also possible to drag the selection with button 1. On button 1, the listbox will also take focus if it has a *normal* state and *-takefocus* is true.

If the selection mode is *multiple* or *extended*, any number of elements may be selected at once, including discontiguous ranges. In *multiple* mode, clicking button 1 on an element toggles its selection state without affecting any other elements. In *extended* mode, pressing button 1 on an element selects it, deselects everything else, and sets the anchor to the element under the mouse; dragging the mouse with button 1 down
extends the selection to include all the elements between the anchor and the element under the mouse, inclusive.

Most people will probably want to use browse mode for single selections and extended mode for multiple selections; the other modes appear to be useful only in special situations.

Any time the selection changes in the listbox, the virtual event <<ListboxSelect>> will be generated. It is easiest to bind to this event to be made aware of any changes to listbox selection.

In addition to the above behavior, the following additional behavior is defined by the default bindings:

[1] In extended mode, the selected range can be adjusted by pressing button 1 with the Shift key down: this modifies the selection to consist of the elements between the anchor and the element under the mouse, inclusive. The un-anchored end of this new selection can also be dragged with the button down.

[2] In extended mode, pressing button 1 with the Control key down starts a toggle operation: the anchor is set to the element under the mouse, and its selection state is reversed. The selection state of other elements is not changed. If the mouse is dragged with button 1 down, then the selection state of all elements between the anchor and the element under the mouse is set to match that of the anchor element; the selection state of all other elements remains what it was before the toggle operation began.

[3] If the mouse leaves the listbox window with button 1 down, the window scrolls away from the mouse, making information visible that used to be off-screen on the side of the mouse. The scrolling continues until the mouse re-enters the window, the button is released, or the end of the listbox is reached.
Mouse button 2 may be used for scanning. If it is pressed and dragged over the listbox, the contents of the listbox drag at high speed in the direction the mouse moves.

If the Up or Down key is pressed, the location cursor (active element) moves up or down one element. If the selection mode is **browse** or **extended** then the new active element is also selected and all other elements are deselected. In **extended** mode the new active element becomes the selection anchor.

In **extended** mode, Shift-Up and Shift-Down move the location cursor (active element) up or down one element and also extend the selection to that element in a fashion similar to dragging with mouse button 1.

The Left and Right keys scroll the listbox view left and right by the width of the character 0. Control-Left and Control-Right scroll the listbox view left and right by the width of the window. Control-Prior and Control-Next also scroll left and right by the width of the window.

The Prior and Next keys scroll the listbox view up and down by one page (the height of the window).

The Home and End keys scroll the listbox horizontally to the left and right edges, respectively.

Control-Home sets the location cursor to the first element in the listbox, selects that element, and deselects everything else in the listbox.
Control-End sets the location cursor to the last element in the listbox, selects that element, and deselects everything else in the listbox.

In extended mode, Control-Shift-Home extends the selection to the first element in the listbox and Control-Shift-End extends the selection to the last element.

In multiple mode, Control-Shift-Home moves the location cursor to the first element in the listbox and Control-Shift-End moves the location cursor to the last element.

The space and Select keys make a selection at the location cursor (active element) just as if mouse button 1 had been pressed over this element.

In extended mode, Control-Shift-space and Shift-Select extend the selection to the active element just as if button 1 had been pressed with the Shift key down.

In extended mode, the Escape key cancels the most recent selection and restores all the elements in the selected range to their previous selection state.

Control-slash selects everything in the widget, except in single and browse modes, in which case it selects the active element and deselects everything else.

Control-backslash deselects everything in the widget, except in browse mode where it has no effect.
The F16 key (labelled Copy on many Sun workstations) or Meta-w copies the selection in the widget to the clipboard, if there is a selection.

The behavior of listboxes can be changed by defining new bindings for individual widgets or by redefining the class bindings.

SEE ALSO

ttk_listbox

KEYWORDS

listbox, widget

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME
scrollbar - Create and manipulate scrollbar widgets

SYNOPSIS

STANDARD OPTIONS
- -activebackground, activeBackground, Foreground
- -background or -bg, background, Background
- -borderwidth or -bd, borderWidth, BorderWidth
- -cursor, cursor, Cursor
- -highlightbackground, highlightBackground,
  HighlightBackground
- -highlightcolor, highlightColor, HighlightColor
- -highlightthickness, highlightThickness, HighlightThickness
- -jump, jump, Jump
- -orient, orient, Orient
- -relief, relief, Relief
- -repeatdelay, repeatDelay, RepeatDelay
- -repeatinterval, repeatInterval, RepeatInterval
- -takefocus, takeFocus, TakeFocus
- -troughcolor, troughColor, Background

WIDGET-SPECIFIC OPTIONS
- -active relieve, activeRelief, ActiveRelief
- -command, command, Command
- -elementborderwidth, elementBorderWidth, BorderWidth
- -width, width, Width

DESCRIPTION

ELEMENTS
arrow1
trough1
slider
trough2
arrow2
WIDGET COMMAND

```
pathname activate ?element?
pathname cget option
pathname configure ?option? ?value option value ...?
pathname delta deltaX deltaY
pathname fraction x y
pathname get
pathname identify x y
pathname set first last
```

SCROLLING COMMANDS

```
prefix moveto fraction
prefix scroll number units
prefix scroll number pages
```

OLD COMMAND SYNTAX

```
pathname set totalUnits windowUnits firstUnit lastUnit
prefix unit
```

BINDINGS
EXAMPLE
SEE ALSO
KEYWORDS

<table>
<thead>
<tr>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrollbar - Create and manipulate scrollbar widgets</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYNOPSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrollbar pathname ?options?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STANDARD OPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>-activebackground, activeBackground, Foreground</td>
</tr>
<tr>
<td>-background or -bg, background, Background</td>
</tr>
<tr>
<td>-borderwidth or -bd, borderWidth, BorderWidth</td>
</tr>
<tr>
<td>-cursor, cursor, Cursor</td>
</tr>
<tr>
<td>-highlightbackground, highlightBackground, HighlightBackground</td>
</tr>
<tr>
<td>-highlightcolor, highlightColor, HighlightColor</td>
</tr>
<tr>
<td>-highlightthickness, highlightThickness, HighlightThickness</td>
</tr>
</tbody>
</table>
-jump, jump, Jump
-orient, orient, Orient
-relief, relief, Relief
-repeatdelay, repeatDelay, RepeatDelay
-repeatinterval, repeatInterval, RepeatInterval
-takefocus, takeFocus, TakeFocus
-troughcolor, troughColor, Background

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: `-activerelief`
Database Name: `activeRelief`
Database Class: `ActiveRelief`
   Specifies the relief to use when displaying the element that is active, if any. Elements other than the active element are always displayed with a raised relief.

Command-Line Name: `-command`
Database Name: `command`
Database Class: `Command`
   Specifies the prefix of a Tcl command to invoke to change the view in the widget associated with the scrollbar. When a user requests a view change by manipulating the scrollbar, a Tcl command is invoked. The actual command consists of this option followed by additional information as described later. This option almost always has a value such as `.t xview` or `.t yview`, consisting of the name of a widget and either `xview` (if the scrollbar is for horizontal scrolling) or `yview` (for vertical scrolling). All scrollable widgets have `xview` and `yview` commands that take exactly the additional arguments appended by the scrollbar as described in **SCROLLING COMMANDS** below.

Command-Line Name: `-elementborderwidth`
Database Name: `elementBorderWidth`
Database Class: `BorderWidth`
   Specifies the width of borders drawn around the internal elements of the scrollbar (the two arrows and the slider). The value may have any of the forms acceptable to `Tk_GetPixels`. If this value is less
than zero, the value of the `borderWidth` option is used in its place.

Command-Line Name: `-width`
Database Name: `width`
Database Class: `Width`

Specifies the desired narrow dimension of the scrollbar window, not including 3-D border, if any. For vertical scrollbars this will be the width and for horizontal scrollbars this will be the height. The value may have any of the forms acceptable to `Tk_GetPixels`.

**DESCRIPTION**

The `scrollbar` command creates a new window (given by the `pathName` argument) and makes it into a scrollbar widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the scrollbar such as its colors, orientation, and relief. The `scrollbar` command returns its `pathName` argument. At the time this command is invoked, there must not exist a window named `pathName`, but `pathName`'s parent must exist.

A scrollbar is a widget that displays two arrows, one at each end of the scrollbar, and a `slider` in the middle portion of the scrollbar. It provides information about what is visible in an associated window that displays a document of some sort (such as a file being edited or a drawing). The position and size of the slider indicate which portion of the document is visible in the associated window. For example, if the slider in a vertical scrollbar covers the top third of the area between the two arrows, it means that the associated window displays the top third of its document.

Scrollbars can be used to adjust the view in the associated window by clicking or dragging with the mouse. See the **BINDINGS** section below for details.

**ELEMENTS**

A scrollbar displays five elements, which are referred to in the widget commands for the scrollbar:
arrow1
   The top or left arrow in the scrollbar.

trough1
   The region between the slider and arrow1.

slider
   The rectangle that indicates what is visible in the associated widget.

trough2
   The region between the slider and arrow2.

arrow2
   The bottom or right arrow in the scrollbar.

**WIDGET COMMAND**

The scrollbar command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The following commands are possible for scrollbar widgets:

*pathName activate ?element?*
   Marks the element indicated by *element* as active, which causes it to be displayed as specified by the `activeBackground` and `activeRelief` options. The only element values understood by this command are `arrow1`, `slider`, or `arrow2`. If any other value is specified then no element of the scrollbar will be active. If *element* is not specified, the command returns the name of the element that is currently active, or an empty string if no element is active.

*pathName cget option*
Returns the current value of the configuration option given by
*option*. *Option* may have any of the values accepted by the
*scrollbar* command.

*pathname configure* ?*option*? ?*value* *option* value ...?
Query or modify the configuration options of the widget. If no *option*
is specified, returns a list describing all of the available options for
*pathname* (see *Tk_ConfigureInfo* for information on the format of
this list). If *option* is specified with no *value*, then the command
returns a list describing the one named option (this list will be
identical to the corresponding sublist of the value returned if no
*option* is specified). If one or more *option*-value pairs are specified,
then the command modifies the given widget option(s) to have the
given value(s); in this case the command returns an empty string.
*Option* may have any of the values accepted by the *scrollbar*
command.

*pathname delta* *deltaX* *deltaY*
Returns a real number indicating the fractional change in the
scrollbar setting that corresponds to a given change in slider
position. For example, if the scrollbar is horizontal, the result
indicates how much the scrollbar setting must change to move the
slider *deltaX* pixels to the right (*deltaY* is ignored in this case). If the
scrollbar is vertical, the result indicates how much the scrollbar
setting must change to move the slider *deltaY* pixels down. The
arguments and the result may be zero or negative.

*pathname fraction* *x* *y*
Returns a real number between 0 and 1 indicating where the point
given by *x* and *y* lies in the trough area of the scrollbar. The value 0
corresponds to the top or left of the trough, the value 1 corresponds
to the bottom or right, 0.5 corresponds to the middle, and so on. *x*
and *y* must be pixel coordinates relative to the scrollbar widget. If *x*
and *y* refer to a point outside the trough, the closest point in the
trough is used.

*pathname get*
Returns the scrollbar settings in the form of a list whose elements
are the arguments to the most recent `set` widget command.

`pathName identify x y`
Returns the name of the element under the point given by `x` and `y` (such as `arrow1`), or an empty string if the point does not lie in any element of the scrollbar. `X` and `y` must be pixel coordinates relative to the scrollbar widget.

`pathName set first last`
This command is invoked by the scrollbar's associated widget to tell the scrollbar about the current view in the widget. The command takes two arguments, each of which is a real fraction between 0 and 1. The fractions describe the range of the document that is visible in the associated widget. For example, if `first` is 0.2 and `last` is 0.4, it means that the first part of the document visible in the window is 20% of the way through the document, and the last visible part is 40% of the way through.

**SCROLLING COMMANDS**

When the user interacts with the scrollbar, for example by dragging the slider, the scrollbar notifies the associated widget that it must change its view. The scrollbar makes the notification by evaluating a Tcl command generated from the scrollbar's `-command` option. The command may take any of the following forms. In each case, `prefix` is the contents of the `-command` option, which usually has a form like `.t yview`

`prefix moveto fraction`
`Fraction` is a real number between 0 and 1. The widget should adjust its view so that the point given by `fraction` appears at the beginning of the widget. If `fraction` is 0 it refers to the beginning of the document. 1.0 refers to the end of the document, 0.333 refers to a point one-third of the way through the document, and so on.

`prefix scroll number units`
The widget should adjust its view by `number` units. The units are defined in whatever way makes sense for the widget, such as characters or lines in a text widget. `Number` is either 1, which
means one unit should scroll off the top or left of the window, or -1, which means that one unit should scroll off the bottom or right of the window.

*prefix* **scroll number pages**

The widget should adjust its view by *number* pages. It is up to the widget to define the meaning of a page; typically it is slightly less than what fits in the window, so that there is a slight overlap between the old and new views. *Number* is either 1, which means the next page should become visible, or -1, which means that the previous page should become visible.

### OLD COMMAND SYNTAX

In versions of Tk before 4.0, the *set* and *get* widget commands used a different form. This form is still supported for backward compatibility, but it is deprecated. In the old command syntax, the *set* widget command has the following form:

```
pathName set totalUnits windowUnits firstUnit lastUnit
```

In this form the arguments are all integers. *TotalUnits* gives the total size of the object being displayed in the associated widget. The meaning of one unit depends on the associated widget; for example, in a text editor widget units might correspond to lines of text. *WindowUnits* indicates the total number of units that can fit in the associated window at one time. *FirstUnit* and *lastUnit* give the indices of the first and last units currently visible in the associated window (zero corresponds to the first unit of the object).

Under the old syntax the *get* widget command returns a list of four integers, consisting of the *totalUnits, windowUnits, firstUnit*, and *lastUnit* values from the last *set* widget command.

The commands generated by scrollbars also have a different form when the old syntax is being used:

```
prefix unit
   Unit is an integer that indicates what should appear at the top or
left of the associated widget’s window. It has the same meaning as the `firstUnit` and `lastUnit` arguments to the `set` widget command.

The most recent `set` widget command determines whether or not to use the old syntax. If it is given two real arguments then the new syntax will be used in the future, and if it is given four integer arguments then the old syntax will be used.

**BINDINGS**

Tk automatically creates class bindings for scrollbars that give them the following default behavior. If the behavior is different for vertical and horizontal scrollbars, the horizontal behavior is described in parentheses.

[1] Pressing button 1 over `arrow1` causes the view in the associated widget to shift up (left) by one unit so that the document appears to move down (right) one unit. If the button is held down, the action auto-repeats.

[2] Pressing button 1 over `trough1` causes the view in the associated widget to shift up (left) by one screenful so that the document appears to move down (right) one screenful. If the button is held down, the action auto-repeats.

[3] Pressing button 1 over the slider and dragging causes the view to drag with the slider. If the `jump` option is true, then the view does not drag along with the slider; it changes only when the mouse button is released.

[4] Pressing button 1 over `trough2` causes the view in the associated widget to shift down (right) by one screenful so that the document appears to move up (left) one screenful. If the button is held down, the action auto-repeats.
Pressing button 1 over arrow2 causes the view in the associated widget to shift down (right) by one unit so that the document appears to move up (left) one unit. If the button is held down, the action auto-repeats.

If button 2 is pressed over the trough or the slider, it sets the view to correspond to the mouse position; dragging the mouse with button 2 down causes the view to drag with the mouse. If button 2 is pressed over one of the arrows, it causes the same behavior as pressing button 1.

If button 1 is pressed with the Control key down, then if the mouse is over arrow1 or trough1 the view changes to the very top (left) of the document; if the mouse is over arrow2 or trough2 the view changes to the very bottom (right) of the document; if the mouse is anywhere else then the button press has no effect.

In vertical scrollbars the Up and Down keys have the same behavior as mouse clicks over arrow1 and arrow2, respectively. In horizontal scrollbars these keys have no effect.

In vertical scrollbars Control-Up and Control-Down have the same behavior as mouse clicks over trough1 and trough2, respectively. In horizontal scrollbars these keys have no effect.

In horizontal scrollbars the Up and Down keys have the same behavior as mouse clicks over arrow1 and arrow2, respectively. In vertical scrollbars these keys have no effect.

In horizontal scrollbars Control-Up and Control-Down have the same behavior as mouse clicks over trough1 and trough2,
respectively. In vertical scrollbars these keys have no effect.

[12] The Prior and Next keys have the same behavior as mouse clicks over `trough1` and `trough2`, respectively.

[13] The Home key adjusts the view to the top (left edge) of the document.

[14] The End key adjusts the view to the bottom (right edge) of the document.

**EXAMPLE**

Create a window with a scrollable `text` widget:

```python
toplevel .tl
  text .tl.t -yscrollcommand {.tl.s set}
  scrollbar .tl.s -command {.tl.t yview}
  grid .tl.t .tl.s -sticky nsew
  grid columnconfigure .tl 0 -weight 1
  grid rowconfigure .tl 0 -weight 1
```

**SEE ALSO**

`ttk:scrollbar`

**KEYWORDS**

`scrollbar`, `widget`
NAME
menu, tk_menuSetFocus - Create and manipulate menu widgets

SYNOPSIS

STANDARD OPTIONS
-activebackground, activeBackground, Foreground
-activeborderwidth, activeBorderWidth, BorderWidth
-activeforeground, activeForeground, Background
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS
-postcommand, postCommand, Command
-selectcolor, selectColor, Background
-tearoff, tearOff, TearOff
-tearoffcommand, tearOffCommand, TearOffCommand
-title, title, Title
-type, type, Type

INTRODUCTION

TYPES OF ENTRIES
COMMAND ENTRIES
SEPARATOR ENTRIES
CHECKBUTTON ENTRIES
RADIOBUTTON ENTRIES
CASCADE ENTRIES
TEAR-OFF ENTRIES
MENUBARS
SPECIAL MENUS IN MENUBARS
CLONES
WIDGET COMMAND

number
active
end
last
none
@number
pattern
pathName activate index
pathName add type ?option value option value ...?
-activebackground value
-activeforeground value
-accelerator value
-background value
-bitmap value
-columnbreak value
-command value
-compound value
-font value
-foreground value
-hidemargin value
-image value
-indicatoron value
-label value
-menu value
-offvalue value
-onvalue value
-selectcolor value
-selectimage value
-state value
-underline value
-value value
-variable value

pathName cget option
**NAME**

menu, tk_menuSetFocus - Create and manipulate menu widgets

**SYNOPSIS**

```python
menu pathName ?options?
tk_menuSetFocus pathName
```

**STANDARD OPTIONS**

- `-activebackground`, `activeBackground`, `Foreground`
- `-activeborderwidth`, `activeBorderWidth`, `BorderWidth`
-activeforeground, activeForeground, Background
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: **-postcommand**
Database Name: **postCommand**
Database Class: **Command**

If this option is specified then it provides a Tcl command to execute each time the menu is posted. The command is invoked by the **post** widget command before posting the menu. Note that in Tk 8.0 on Macintosh and Windows, all post-commands in a system of menus are executed before any of those menus are posted. This is due to the limitations in the individual platforms' menu managers.

Command-Line Name: **-selectcolor**
Database Name: **selectColor**
Database Class: **BackgroundColor**

For menu entries that are check buttons or radio buttons, this option specifies the color to display in the indicator when the check button or radio button is selected.

Command-Line Name: **-tearoff**
Database Name: **tearOff**
Database Class: **TearOff**

This option must have a proper boolean value, which specifies whether or not the menu should include a tear-off entry at the top. If so, it will exist as entry 0 of the menu and the other entries will number starting at 1. The default menu bindings arrange for the menu to be torn off when the tear-off entry is invoked.
The menu command creates a new top-level window (given by the \textit{pathName} argument) and makes it into a menu widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the menu such as its colors and font. The menu command returns its \textit{pathName} argument. At the time this command is invoked, there must not exist a window named \textit{pathName}, but \textit{pathName}'s parent must exist.
A menu is a widget that displays a collection of one-line entries arranged in one or more columns. There exist several different types of entries, each with different properties. Entries of different types may be combined in a single menu. Menu entries are not the same as entry widgets. In fact, menu entries are not even distinct widgets; the entire menu is one widget.

Menu entries are displayed with up to three separate fields. The main field is a label in the form of a text string, a bitmap, or an image, controlled by the -label, -bitmap, and -image options for the entry. If the -accelerator option is specified for an entry then a second textual field is displayed to the right of the label. The accelerator typically describes a keystroke sequence that may be typed in the application to cause the same result as invoking the menu entry. The third field is an indicator. The indicator is present only for checkbutton or radiobutton entries. It indicates whether the entry is selected or not, and is displayed to the left of the entry's string.

In normal use, an entry becomes active (displays itself differently) whenever the mouse pointer is over the entry. If a mouse button is released over the entry then the entry is invoked. The effect of invocation is different for each type of entry; these effects are described below in the sections on individual entries.

Entries may be disabled, which causes their labels and accelerators to be displayed with dimmer colors. The default menu bindings will not allow a disabled entry to be activated or invoked. Disabled entries may be re-enabled, at which point it becomes possible to activate and invoke them again.

Whenever a menu's active entry is changed, a <<MenuSelect>> virtual event is send to the menu. The active item can then be queried from the menu, and an action can be taken, such as setting context-sensitive help text for the entry.

### TYPES OF ENTRIES

#### COMMAND ENTRIES
The most common kind of menu entry is a command entry, which behaves much like a button widget. When a command entry is invoked, a Tcl command is executed. The Tcl command is specified with the -command option.

**SEPARATOR ENTRIES**

A separator is an entry that is displayed as a horizontal dividing line. A separator may not be activated or invoked, and it has no behavior other than its display appearance.

**CHECKBUTTON ENTRIES**

A checkbutton menu entry behaves much like a checkbutton widget. When it is invoked it toggles back and forth between the selected and deselected states. When the entry is selected, a particular value is stored in a particular global variable (as determined by the -onvalue and -variable options for the entry); when the entry is deselected another value (determined by the -offvalue option) is stored in the global variable. An indicator box is displayed to the left of the label in a checkbutton entry. If the entry is selected then the indicator's center is displayed in the color given by the -selectcolor option for the entry; otherwise the indicator's center is displayed in the background color for the menu. If a -command option is specified for a checkbutton entry, then its value is evaluated as a Tcl command each time the entry is invoked; this happens after toggling the entry's selected state.

**RADIOBUTTON ENTRIES**

A radiobutton menu entry behaves much like a radiobutton widget. Radiobutton entries are organized in groups of which only one entry may be selected at a time. Whenever a particular entry becomes selected it stores a particular value into a particular global variable (as determined by the -value and -variable options for the entry). This action causes any previously-selected entry in the same group to deselect itself. Once an entry has become selected, any change to the entry's associated variable will cause the entry to deselect itself.
Grouping of radiobutton entries is determined by their associated variables: if two entries have the same associated variable then they are in the same group. An indicator diamond is displayed to the left of the label in each radiobutton entry. If the entry is selected then the indicator's center is displayed in the color given by the `-selectcolor` option for the entry; otherwise the indicator's center is displayed in the background color for the menu. If a `-command` option is specified for a radiobutton entry, then its value is evaluated as a Tcl command each time the entry is invoked; this happens after selecting the entry.

**CASCADE ENTRIES**

A cascade entry is one with an associated menu (determined by the `-menu` option). Cascade entries allow the construction of cascading menus. The `postcascade` widget command can be used to post and unpost the associated menu just next to of the cascade entry. The associated menu must be a child of the menu containing the cascade entry (this is needed in order for menu traversal to work correctly).

A cascade entry posts its associated menu by invoking a Tcl command of the form

```
menu post x y
```

where `menu` is the path name of the associated menu, and `x` and `y` are the root-window coordinates of the upper-right corner of the cascade entry. On Unix, the lower-level menu is unposted by executing a Tcl command with the form

```
menu unpost
```

where `menu` is the name of the associated menu. On other platforms, the platform's native code takes care of unposting the menu.
If a `-command` option is specified for a cascade entry then it is evaluated as a Tcl command whenever the entry is invoked. This is not supported on Windows.

**TEAR-OFF ENTRIES**

A tear-off entry appears at the top of the menu if enabled with the `tearOff` option. It is not like other menu entries in that it cannot be created with the `add` widget command and cannot be deleted with the `delete` widget command. When a tear-off entry is created it appears as a dashed line at the top of the menu. Under the default bindings, invoking the tear-off entry causes a torn-off copy to be made of the menu and all of its submenus.

**MENUBARS**

Any menu can be set as a menubar for a toplevel window (see `toplevel` command for syntax). On the Macintosh, whenever the toplevel is in front, this menu's cascade items will appear in the menubar across the top of the main monitor. On Windows and Unix, this menu's items will be displayed in a menubar across the top of the window. These menus will behave according to the interface guidelines of their platforms. For every menu set as a menubar, a clone menu is made. See the `CLONES` section for more information.

As noted, menubars may behave differently on different platforms. One example of this concerns the handling of checkbuttons and radiobuttons within the menu. While it is permitted to put these menu elements on menubars, they may not be drawn with indicators on some platforms, due to system restrictions.

**SPECIAL MENUS IN MENUBARS**

Certain menus in a menubar will be treated specially. On the Macintosh, access to the special Application and Help menus is provided. On Windows, access to the Windows System menu in each window is provided. On X Windows, a special right-justified help menu is provided.
In all cases, these menus must be created with the command name of the menubar menu concatenated with the special name. So for a menubar named .menubar, on the Macintosh, the special menus would be .menubar.apple and .menubar.help; on Windows, the special menu would be .menubar.system; on X Windows, the help menu would be .menubar.help.

When Tk sees a .menubar.apple menu on the Macintosh, that menu's contents make up the first items of the Application menu whenever the window containing the menubar is in front. After all of the Tk-defined items, the menu will have a separator, followed by all standard Application menu items.

When Tk sees a Help menu on the Macintosh, the menu's contents are appended to the standard Help menu on the right of the user's menubar whenever the window's menubar is in front. The first items in the menu are provided by Mac OS X.

When Tk sees a System menu on Windows, its items are appended to the system menu that the menubar is attached to. This menu has an icon representing a spacebar, and can be invoked with the mouse or by typing Alt+Spacebar. Due to limitations in the Windows API, any font changes, colors, images, bitmaps, or tearoff images will not appear in the system menu.

When Tk see a Help menu on X Windows, the menu is moved to be last in the menubar and is right justified.

**CLONES**

When a menu is set as a menubar for a toplevel window, or when a menu is torn off, a clone of the menu is made. This clone is a menu widget in its own right, but it is a child of the original. Changes in the configuration of the original are reflected in the clone. Additionally, any cascades that are pointed to are also cloned so that menu traversal will work right. Clones are destroyed when either the tearoff or menubar goes away, or when the original menu is destroyed.
The **menu** command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command.

Many of the widget commands for a menu take as one argument an indicator of which entry of the menu to operate on. These indicators are called *indexes* and may be specified in any of the following forms:

- **number**
  - Specifies the entry numerically, where 0 corresponds to the top-most entry of the menu, 1 to the entry below it, and so on.

- **active**
  - Indicates the entry that is currently active. If no entry is active then this form is equivalent to `none`. This form may not be abbreviated.

- **end**
  - Indicates the bottommost entry in the menu. If there are no entries in the menu then this form is equivalent to `none`. This form may not be abbreviated.

- **last**
  - Same as `end`.

- **none**
  - Indicates “no entry at all”; this is used most commonly with the **activate** option to deactivate all the entries in the menu. In most cases the specification of `none` causes nothing to happen in the widget command. This form may not be abbreviated.

- **@number**
In this form, number is treated as a y-coordinate in the menu's window; the entry closest to that y-coordinate is used. For example, “@0” indicates the top-most entry in the window.

**pattern**
If the index does not satisfy one of the above forms then this form is used. Pattern is pattern-matched against the label of each entry in the menu, in order from the top down, until a matching entry is found. The rules of [Tcl StringMatch](#) are used.

The following widget commands are possible for menu widgets:

**pathName activate index**
Change the state of the entry indicated by index to active and redisplay it using its active colors. Any previously-active entry is deactivated. If index is specified as none, or if the specified entry is disabled, then the menu ends up with no active entry. Returns an empty string.

**pathName add type ?option value option value ...?**
Add a new entry to the bottom of the menu. The new entry's type is given by type and must be one of cascade, checkbutton, command, radiobutton, or separator, or a unique abbreviation of one of the above. If additional arguments are present, they specify any of the following options:

**-activebackground value**
Specifies a background color to use for displaying this entry when it is active. If this option is specified as an empty string (the default), then the activeBackground option for the overall menu is used. If the tk_strictMotif variable has been set to request strict Motif compliance, then this option is ignored and the -background option is used in its place. This option is not available for separator or tear-off entries.

**-activeforeground value**
Specifies a foreground color to use for displaying this entry when it is active. If this option is specified as an empty string
(the default), then the **activeForeground** option for the overall menu is used. This option is not available for separator or tear-off entries.

**-accelerator** `value`
Specifies a string to display at the right side of the menu entry. Normally describes an accelerator keystroke sequence that may be typed to invoke the same function as the menu entry. This option is not available for separator or tear-off entries.

**-background** `value`
Specifies a background color to use for displaying this entry when it is in the normal state (neither active nor disabled). If this option is specified as an empty string (the default), then the **background** option for the overall menu is used. This option is not available for separator or tear-off entries.

**-bitmap** `value`
Specifies a bitmap to display in the menu instead of a textual label, in any of the forms accepted by `Tk_GetBitmap`. This option overrides the **-label** option (as controlled by the **compound** option) but may be reset to an empty string to enable a textual label to be displayed. If a **-image** option has been specified, it overrides **-bitmap**. This option is not available for separator or tear-off entries.

**-columnbreak** `value`
When this option is zero, the entry appears below the previous entry. When this option is one, the entry appears at the top of a new column in the menu.

**-command** `value`
Specifies a Tcl command to execute when the menu entry is invoked. Not available for separator or tear-off entries.

**-compound** `value`
Specifies whether the menu entry should display both an image and text, and if so, where the image should be placed
relative to the text. Valid values for this option are **bottom**, **center**, **left**, **none**, **right** and **top**. The default value is **none**, meaning that the button will display either an image or text, depending on the values of the **-image** and **-bitmap** options.

**-font** value
Specifies the font to use when drawing the label or accelerator string in this entry. If this option is specified as an empty string (the default) then the **font** option for the overall menu is used. This option is not available for separator or tear-off entries.

**-foreground** value
Specifies a foreground color to use for displaying this entry when it is in the normal state (neither active nor disabled). If this option is specified as an empty string (the default), then the **foreground** option for the overall menu is used. This option is not available for separator or tear-off entries.

**-hidemargin** value
Specifies whether the standard margins should be drawn for this menu entry. This is useful when creating palette with images in them, i.e., color palettes, pattern palettes, etc. 1 indicates that the margin for the entry is hidden; 0 means that the margin is used.

**-image** value
Specifies an image to display in the menu instead of a text string or bitmap. The image must have been created by some previous invocation of **image create**. This option overrides the **-label** and **-bitmap** options (as controlled by the **-compound** option) but may be reset to an empty string to enable a textual or bitmap label to be displayed. This option is not available for separator or tear-off entries.

**-indicatoron** value
Available only for checkbutton and radiobutton entries. **Value** is a boolean that determines whether or not the indicator should be displayed.
-**label** *value*
  Specifies a string to display as an identifying label in the menu entry. Not available for separator or tear-off entries.

-**menu** *value*
  Available only for cascade entries. Specifies the path name of the submenu associated with this entry. The submenu must be a child of the menu.

-**offvalue** *value*
  Available only for checkbutton entries. Specifies the value to store in the entry's associated variable when the entry is deselected.

-**onvalue** *value*
  Available only for checkbutton entries. Specifies the value to store in the entry's associated variable when the entry is selected.

-**selectcolor** *value*
  Available only for checkbutton and radiobutton entries. Specifies the color to display in the indicator when the entry is selected. If the value is an empty string (the default) then the **selectColor** option for the menu determines the indicator color.

-**selectimage** *value*
  Available only for checkbutton and radiobutton entries. Specifies an image to display in the entry (in place of the **image** option) when it is selected. *Value* is the name of an image, which must have been created by some previous invocation of **image create**. This option is ignored unless the **image** option has been specified.

-**state** *value*
  Specifies one of three states for the entry: **normal**, **active**, or **disabled**. In normal state the entry is displayed using the **foreground** option for the menu and the **background** option
from the entry or the menu. The active state is typically used when the pointer is over the entry. In active state the entry is displayed using the \texttt{activeForeground} option for the menu along with the \texttt{activebackground} option from the entry. Disabled state means that the entry should be insensitive: the default bindings will refuse to activate or invoke the entry. In this state the entry is displayed according to the \texttt{disabledForeground} option for the menu and the \texttt{background} option from the entry. This option is not available for separator entries.

- \texttt{underline value}
  Specifies the integer index of a character to underline in the entry. This option is also queried by the default bindings and used to implement keyboard traversal. 0 corresponds to the first character of the text displayed in the entry, 1 to the next character, and so on. If a bitmap or image is displayed in the entry then this option is ignored. This option is not available for separator or tear-off entries.

- \texttt{value value}
  Available only for radiobutton entries. Specifies the value to store in the entry's associated variable when the entry is selected. If an empty string is specified, then the \texttt{-label} option for the entry as the value to store in the variable.

- \texttt{variable value}
  Available only for checkbutton and radiobutton entries. Specifies the name of a global value to set when the entry is selected. For checkbutton entries the variable is also set when the entry is deselected. For radiobutton entries, changing the variable causes the currently-selected entry to deselect itself.

The \texttt{add} widget command returns an empty string.

\texttt{pathName cget option}
Returns the current value of the configuration option given by \texttt{option}. \texttt{Option} may have any of the values accepted by the \texttt{menu}
pathName clone newPathname ?cloneType? 
Makes a clone of the current menu named newPathName. This clone is a menu in its own right, but any changes to the clone are propagated to the original menu and vice versa. cloneType can be normal, menubar, or tearoff. Should not normally be called outside of the Tk library. See the CLONES section for more information.

pathName configure ?option? ?value option value ...?
Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option-value pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. Option may have any of the values accepted by the menu command.

pathName delete index1 ?index2?
Delete all of the menu entries between index1 and index2 inclusive. If index2 is omitted then it defaults to index1. Attempts to delete a tear-off menu entry are ignored (instead, you should change the tearOff option to remove the tear-off entry).

pathName entrycget index option
Returns the current value of a configuration option for the entry given by index. Option may have any of the values accepted by the add widget command.

pathName entryconfigure index ?options?
This command is similar to the configure command, except that it applies to the options for an individual entry, whereas configure applies to the options for the menu as a whole. Options may have
any of the values accepted by the add widget command. If options are specified, options are modified as indicated in the command and the command returns an empty string. If no options are specified, returns a list describing the current options for entry index (see Tk ConfigureInfo for information on the format of this list).

pathName index index
Returns the numerical index corresponding to index, or none if index was specified as none.

pathName insert index type ?option value option value ...?
Same as the add widget command except that it inserts the new entry just before the entry given by index, instead of appending to the end of the menu. The type, option, and value arguments have the same interpretation as for the add widget command. It is not possible to insert new menu entries before the tear-off entry, if the menu has one.

pathName invoke index
Invoke the action of the menu entry. See the sections on the individual entries above for details on what happens. If the menu entry is disabled then nothing happens. If the entry has a command associated with it then the result of that command is returned as the result of the invoke widget command. Otherwise the result is an empty string. Note: invoking a menu entry does not automatically unpost the menu; the default bindings normally take care of this before invoking the invoke widget command.

pathName post x y
Arrange for the menu to be displayed on the screen at the root-window coordinates given by x and y. These coordinates are adjusted if necessary to guarantee that the entire menu is visible on the screen. This command normally returns an empty string. If the postCommand option has been specified, then its value is executed as a Tcl script before posting the menu and the result of that script is returned as the result of the post widget command. If an error returns while executing the command, then the error is
returned without posting the menu.

`pathName postcascade index`
Posts the submenu associated with the cascade entry given by `index`, and unposts any previously posted submenu. If `index` does not correspond to a cascade entry, or if `pathName` is not posted, the command has no effect except to unpost any currently posted submenu.

`pathName type index`
Returns the type of the menu entry given by `index`. This is the `type` argument passed to the `add` widget command when the entry was created, such as `command` or `separator`, or `tearoff` for a tear-off entry.

`pathName unpost`
Unmap the window so that it is no longer displayed. If a lower-level cascaded menu is posted, unpost that menu. Returns an empty string. This subcommand does not work on Windows and the Macintosh, as those platforms have their own way of unposting menus.

`pathName xposition index`
Returns a decimal string giving the x-coordinate within the menu window of the leftmost pixel in the entry specified by `index`.

`pathName yposition index`
Returns a decimal string giving the y-coordinate within the menu window of the topmost pixel in the entry specified by `index`.

**MENU CONFIGURATIONS**

The default bindings support four different ways of using menus:

**Pulldown Menus in Menubar**
This is the most common case. You create a menu widget that will become the menu bar. You then add cascade entries to this menu, specifying the pull down menus you wish to use in your menu bar.
You then create all of the pulldowns. Once you have done this, specify the menu using the `-menu` option of the toplevel's widget command. See the `toplevel` manual entry for details.

**Pulldown Menus in Menu Buttons**
This is the compatible way to do menu bars. You create one menubutton widget for each top-level menu, and typically you arrange a series of menubuttons in a row in a menubar window. You also create the top-level menus and any cascaded submenus, and tie them together with `-menu` options in menubuttons and cascade menu entries. The top-level menu must be a child of the menubutton, and each submenu must be a child of the menu that refers to it. Once you have done this, the default bindings will allow users to traverse and invoke the tree of menus via its menubutton; see the `menubutton` manual entry for details.

**Popup Menus**
Popup menus typically post in response to a mouse button press or keystroke. You create the popup menus and any cascaded submenus, then you call the `tk_popup` procedure at the appropriate time to post the top-level menu.

**Option Menus**
An option menu consists of a menubutton with an associated menu that allows you to select one of several values. The current value is displayed in the menubutton and is also stored in a global variable. Use the `tk_optionMenu` procedure to create option menubuttons and their menus.

**Torn-off Menus**
You create a torn-off menu by invoking the tear-off entry at the top of an existing menu. The default bindings will create a new menu that is a copy of the original menu and leave it permanently posted as a top-level window. The torn-off menu behaves just the same as the original menu.

**DEFAULT BINDINGS**
Tk automatically creates class bindings for menus that give them the following default behavior:

[1] When the mouse enters a menu, the entry underneath the mouse cursor activates; as the mouse moves around the menu, the active entry changes to track the mouse.

[2] When the mouse leaves a menu all of the entries in the menu deactivate, except in the special case where the mouse moves from a menu to a cascaded submenu.

[3] When a button is released over a menu, the active entry (if any) is invoked. The menu also unposts unless it is a torn-off menu.

[4] The Space and Return keys invoke the active entry and unpost the menu.

[5] If any of the entries in a menu have letters underlined with the -underline option, then pressing one of the underlined letters (or its upper-case or lower-case equivalent) invokes that entry and unposts the menu.

[6] The Escape key aborts a menu selection in progress without invoking any entry. It also unposts the menu unless it is a torn-off menu.

[7] The Up and Down keys activate the next higher or lower entry in the menu. When one end of the menu is reached, the active entry wraps around to the other end.

[8]
The Left key moves to the next menu to the left. If the current menu is a cascaded submenu, then the submenu is unposted and the current menu entry becomes the cascade entry in the parent. If the current menu is a top-level menu posted from a menubutton, then the current menubutton is unposted and the next menubutton to the left is posted. Otherwise the key has no effect. The left-right order of menubuttons is determined by their stacking order: Tk assumes that the lowest menubutton (which by default is the first one created) is on the left.

[9]

The Right key moves to the next menu to the right. If the current entry is a cascade entry, then the submenu is posted and the current menu entry becomes the first entry in the submenu. Otherwise, if the current menu was posted from a menubutton, then the current menubutton is unposted and the next menubutton to the right is posted.

Disabled menu entries are non-responsive: they do not activate and they ignore mouse button presses and releases.

Several of the bindings make use of the command `tk_menuSetFocus`. It saves the current focus and sets the focus to its `pathName` argument, which is a menu widget.

The behavior of menus can be changed by defining new bindings for individual widgets or by redefining the class bindings.

**BUGS**

At present it is not possible to use the option database to specify values for the options to individual entries.

**SEE ALSO**

`bind`, `menubutton`, `ttk::menubutton`, `toplevel`

**KEYWORDS**
menu, widget

<table>
<thead>
<tr>
<th>Copyright © 1995-1997 Roger E. Critchlow Jr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copyright © 1990-1994 The Regents of the University of California.</td>
</tr>
<tr>
<td>Copyright © 1994-1997 Sun Microsystems, Inc.</td>
</tr>
</tbody>
</table>
NAME
  ttk::entry - Editable text field widget

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS
  -class
  -cursor, cursor, Cursor
  -style
  -takefocus, takeFocus,TakeFocus
  -xscrollcommand, xScrollCommand, ScrollCommand

WIDGET-SPECIFIC OPTIONS
  -exportselection, exportSelection, ExportSelection
  -invalidcommand, invalidCommand, InvalidCommand
  -justify, justify, Justify
  -show, show, Show
  -state, state, State
  -textvariable, textVariable, Variable
  -validate, validate, Validate
  -validatecommand, validateCommand, ValidateCommand
  -width, width, Width

NOTES

INDICES
  number
  @number
  end
  insert
  sel.first
  sel.last

WIDGET COMMAND
  pathName bbox index
  pathName cget option
pathName configure ?option? ?value option value ...
pathName delete first ?last?
pathName get
pathName icursor index
pathName identify x y
pathName index index
pathName insert index string
pathName instate statespec ?script?
pathName selection option arg
  pathName selection clear
  pathName selection present
  pathName selection range start end
pathName state ?stateSpec?
pathName validate
pathName xview args
  pathName xview
  pathName xview index
  pathName xview moveto fraction
  pathName xview scroll number what

VALIDATION
VALIDATION MODES
  none
  key
  focus
  focusin
  focusout
  all

VALIDATION SCRIPT SUBSTITUTIONS
  %d
  %i
  %P
  %s
  %S
  %v
  %V
  %W

DIFFERENCES FROM TK ENTRY WIDGET VALIDATION
ttk::entry - Editable text field widget

**SYNOPSIS**

**DESCRIPTION**

An ttk::entry widget displays a one-line text string and allows that string to be edited by the user. The value of the string may be linked to a Tcl variable with the `-textvariable` option. Entry widgets support horizontal scrolling with the standard `-xscrollcommand` option and `xview` widget command.

**STANDARD OPTIONS**

- `class`
- `cursor, cursor, Cursor`
- `style`
- `takefocus, takeFocus, TakeFocus`
- `xscrollcommand, xScrollCommand, ScrollCommand`

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: `-exportselection`
Database Name: `exportSelection`
Database Class: `ExportSelection`

A boolean value specifying whether or not a selection in the widget should be linked to the X selection. If the selection is exported, then selecting in the widget deselects the current X selection, selecting outside the widget deselects any widget selection, and the widget
will respond to selection retrieval requests when it has a selection.

Command-Line Name: `-invalidcommand`
Database Name: `invalidCommand`
Database Class: `InvalidCommand`
A script template to evaluate whenever the `validateCommand`
returns 0. See VALIDATION below for more information.

Command-Line Name: `-justify`
Database Name: `justify`
Database Class: `Justify`
Specifies how the text is aligned within the entry widget. One of
left, center, or right.

Command-Line Name: `-show`
Database Name: `show`
Database Class: `Show`
If this option is specified, then the true contents of the entry are not
displayed in the window. Instead, each character in the entry's
value will be displayed as the first character in the value of this
option, such as “*” or a bullet. This is useful, for example, if the
entry is to be used to enter a password. If characters in the entry
are selected and copied elsewhere, the information copied will be
what is displayed, not the true contents of the entry.

Command-Line Name: `-state`
Database Name: `state`
Database Class: `State`
Compatibility option; see `ttk::widget(n)` for details. Specifies one of
three states for the entry, normal, disabled, or readonly. See
WIDGET STATES, below.

Command-Line Name: `-textvariable`
Database Name: `textVariable`
Database Class: `Variable`
Specifies the name of a variable whose value is linked to the entry
widget's contents. Whenever the variable changes value, the
widget's contents are updated, and vice versa.
Command-Line Name: -validate
Database Name: validate
Database Class: Validate
  Specifies the mode in which validation should operate: none, focus, focusin, focusout, key, or all. Default is none, meaning that validation is disabled. See VALIDATION below.

Command-Line Name: -validatecommand
Database Name: validateCommand
Database Class: ValidateCommand
  A script template to evaluate whenever validation is triggered. If set to the empty string (the default), validation is disabled. The script must return a boolean value. See VALIDATION below.

Command-Line Name: -width
Database Name: width
Database Class: Width
  Specifies an integer value indicating the desired width of the entry window, in average-size characters of the widget's font.

NOTES

A portion of the entry may be selected as described below. If an entry is exporting its selection (see the exportSelection option), then it will observe the standard X11 protocols for handling the selection; entry selections are available as type STRING. Entries also observe the standard Tk rules for dealing with the input focus. When an entry has the input focus it displays an insert cursor to indicate where new characters will be inserted.

Entries are capable of displaying strings that are too long to fit entirely within the widget's window. In this case, only a portion of the string will be displayed; commands described below may be used to change the view in the window. Entries use the standard xScrollCommand mechanism for interacting with scrollbars (see the description of the xScrollCommand option for details).
Many of the `entry` widget commands take one or more indices as arguments. An index specifies a particular character in the entry's string, in any of the following ways:

**number**
Specifies the character as a numerical index, where 0 corresponds to the first character in the string.

**@number**
In this form, `number` is treated as an x-coordinate in the entry's window; the character spanning that x-coordinate is used. For example, “@0” indicates the left-most character in the window.

**end**
Indicates the character just after the last one in the entry's string. This is equivalent to specifying a numerical index equal to the length of the entry's string.

**insert**
Indicates the character adjacent to and immediately following the insert cursor.

**sel.first**
Indicates the first character in the selection. It is an error to use this form if the selection is not in the entry window.

**sel.last**
Indicates the character just after the last one in the selection. It is an error to use this form if the selection is not in the entry window.

Abbreviations may be used for any of the forms above, e.g. “e” or “sel.f”. In general, out-of-range indices are automatically rounded to the nearest legal value.

**WIDGET COMMAND**

The following commands are possible for entry widgets:

`pathName bbox index`
Returns a list of four numbers describing the bounding box of the character given by `index`. The first two elements of the list give the x and y coordinates of the upper-left corner of the screen area covered by the character (in pixels relative to the widget) and the last two elements give the width and height of the character, in pixels. The bounding box may refer to a region outside the visible area of the window.

`pathName cget option`
Returns the current value of the specified `option`. See `ttk::widget(n)`.

`pathName configure ?option? ?value option value ...?`
Modify or query widget options. See `ttk::widget(n)`.

`pathName delete first ?!last?`
Delete one or more elements of the entry. `First` is the index of the first character to delete, and `last` is the index of the character just after the last one to delete. If `last` is not specified it defaults to `first+1`, i.e. a single character is deleted. This command returns the empty string.

`pathName get`
Returns the entry's string.

`pathName icursor index`
Arrange for the insert cursor to be displayed just before the character given by `index`. Returns the empty string.

`pathName identify x y`
Returns the name of the element at position `x`, `y`, or the empty string if the coordinates are outside the window.

`pathName index index`
Returns the numerical index corresponding to `index`.

`pathName insert index string`
Insert `string` just before the character indicated by `index`. Returns the empty string.
**pathName instate statespec ?script?**
Test the widget state. See *ttk::widget(n)*.

**pathName selection option arg**
This command is used to adjust the selection within an entry. It has several forms, depending on `option`:

**pathName selection clear**
Clear the selection if it is currently in this widget. If the selection is not in this widget then the command has no effect. Returns the empty string.

**pathName selection present**
Returns 1 if there are characters selected in the entry, 0 if nothing is selected.

**pathName selection range start end**
Sets the selection to include the characters starting with the one indexed by `start` and ending with the one just before `end`. If `end` refers to the same character as `start` or an earlier one, then the entry's selection is cleared.

**pathName state ?stateSpec?**
Modify or query the widget state. See *ttk::widget(n)*.

**pathName validate**
Force revalidation, independent of the conditions specified by the `-validate` option. Returns 0 if validation fails, 1 if it succeeds. Sets or clears the `invalid` state accordingly.

**pathName xview args**
This command is used to query and change the horizontal position of the text in the widget's window. It can take any of the following forms:

**pathName xview**
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal
span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the entry's text is off-screen to the left, the middle 40% is visible in the window, and 40% of the text is off-screen to the right. These are the same values passed to scrollbars via the -xscrollcommand option.

`pathName xview index`  
Adjusts the view in the window so that the character given by `index` is displayed at the left edge of the window.

`pathName xview moveto fraction`  
Adjusts the view in the window so that the character `fraction` of the way through the text appears at the left edge of the window. `Fraction` must be a fraction between 0 and 1.

`pathName xview scroll number what`  
This command shifts the view in the window left or right according to `number` and `what`. `Number` must be an integer. `What` must be either `units` or `pages`. If `what` is `units`, the view adjusts left or right by `number` average-width characters on the display; if it is `pages` then the view adjusts by `number` screenfuls. If `number` is negative then characters farther to the left become visible; if it is positive then characters farther to the right become visible.

**VALIDATION**

The `-validate`, `-validatecommand`, and `-invalidcommand` options are used to enable entry widget validation.

**VALIDATION MODES**

There are two main validation modes: `prevalidation`, in which the `-validatecommand` is evaluated prior to each edit and the return value is used to determine whether to accept or reject the change; and `revalidation`, in which the `-validatecommand` is evaluated to determine whether the current value is valid.
The -validate option determines when validation occurs; it may be set to any of the following values:

**none**
Default. This means validation will only occur when specifically requested by the validate widget command.

**key**
The entry will be prevalidated prior to each edit (specifically, whenever the insert or delete widget commands are called). If prevalidation fails, the edit is rejected.

**focus**
The entry is revalidated when the entry receives or loses focus.

**focusin**
The entry is revalidated when the entry receives focus.

**focusout**
The entry is revalidated when the entry loses focus.

**all**
Validation is performed for all above conditions.

The -invalidcommand is evaluated whenever the -validatecommand returns a false value.

The -validatecommand and -invalidcommand may modify the entry widget's value via the widget insert or delete commands, or by setting the linked -textvariable. If either does so during prevalidation, then the edit is rejected regardless of the value returned by the -validatecommand.

If -validatecommand is empty (the default), validation always succeeds.
It is possible to perform percent substitutions on the `-validatecommand` and `invalidCommand`, just as in a `bind` script. The following substitutions are recognized:

`%d`
Type of action: 1 for `insert` prevalidation, 0 for `delete` prevalidation, or -1 for revalidation.

`%i`
Index of character string to be inserted/deleted, if any, otherwise -1.

`%P`
In prevalidation, the new value of the entry if the edit is accepted. In revalidation, the current value of the entry.

`%s`
The current value of entry prior to editing.

`%S`
The text string being inserted/deleted, if any, `{}` otherwise.

`%v`
The current value of the `-validate` option.

`%V`
The validation condition that triggered the callback (key, focusin, focusout, or forced).

`%W`
The name of the entry widget.

---

**DIFFERENCES FROM TK ENTRY WIDGET VALIDATION**

- The standard Tk entry widget automatically disables validation (by setting `-validate` to `none`) if the `-validatecommand` or `-invalidcommand` modifies the entry's value. The Tk themed entry widget only disables validation if one of the validation scripts raises an error, or if `-validatecommand` does not return a valid boolean.
value. (Thus, it is not necessary to reenable validation after modifying the entry value in a validation script).

- The standard entry widget invokes validation whenever the linked -textvariable is modified; the Tk themed entry widget does not.

### DEFAULT BINDINGS

The entry widget's default bindings enable the following behavior. In the descriptions below, “word” refers to a contiguous group of letters, digits, or “_” characters, or any single character other than these.

- Clicking mouse button 1 positions the insert cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget. Dragging with mouse button 1 down strokes out a selection between the insert cursor and the character under the mouse.

- Double-clicking with mouse button 1 selects the word under the mouse and positions the insert cursor at the end of the word. Dragging after a double click strokes out a selection consisting of whole words.

- Triple-clicking with mouse button 1 selects all of the text in the entry and positions the insert cursor at the end of the line.

- The ends of the selection can be adjusted by dragging with mouse button 1 while the Shift key is down. If the button is double-clicked before dragging then the selection will be adjusted in units of whole words.

- Clicking mouse button 1 with the Control key down will position the insert cursor in the entry without affecting the selection.

- If any normal printing characters are typed in an entry, they are inserted at the point of the insert cursor.

- The view in the entry can be adjusted by dragging with mouse button 2. If mouse button 2 is clicked without moving the mouse,
the selection is copied into the entry at the position of the mouse cursor.

- If the mouse is dragged out of the entry on the left or right sides while button 1 is pressed, the entry will automatically scroll to make more text visible (if there is more text off-screen on the side where the mouse left the window).

- The Left and Right keys move the insert cursor one character to the left or right; they also clear any selection in the entry. If Left or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character. Control-Left and Control-Right move the insert cursor by words, and Control-Shift-Left and Control-Shift-Right move the insert cursor by words and also extend the selection. Control-b and Control-f behave the same as Left and Right, respectively.

- The Home key and Control-a move the insert cursor to the beginning of the entry and clear any selection in the entry. Shift-Home moves the insert cursor to the beginning of the entry and extends the selection to that point.

- The End key and Control-e move the insert cursor to the end of the entry and clear any selection in the entry. Shift-End moves the cursor to the end and extends the selection to that point.

- Control-/ selects all the text in the entry.

- Control-\ clears any selection in the entry.

- The standard Tk <<Cut>>, <<Copy>>, <<Paste>>, and <<Clear>> virtual events operate on the selection in the expected manner.

- The Delete key deletes the selection, if there is one in the entry. If there is no selection, it deletes the character to the right of the insert cursor.

- The BackSpace key and Control-h delete the selection, if there is
one in the entry. If there is no selection, it deletes the character to
the left of the insert cursor.

• Control-d deletes the character to the right of the insert cursor.

• Control-k deletes all the characters to the right of the insertion
cursor.

WIDGET STATES

In the disabled state, the entry cannot be edited and the text cannot be
selected. In the readonly state, no insert cursor is displayed and the
text cannot be edited (specifically: the insert and delete commands
have no effect). The disabled state is the same as readonly, and in
addition text cannot be selected.

Note that changes to the linked -textvariable will still be reflected in the
entry, even if it is disabled or readonly.

Typically, the text is “grayed-out” in the disabled state, and a different
background is used in the readonly state.

The entry widget sets the invalid state if revalidation fails, and clears it
whenever validation succeeds.

SEE ALSO

ttk::widget, entry

KEYWORDS

entry, widget, text field

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 2004 Joe English
NAME

ttk::treeview - hierarchical multicolumn data display widget

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS

-class
-cursor, cursor, Cursor
-style
-takefocus, takeFocus, TakeFocus
-xscrollcommand, xScrollCommand, ScrollCommand
-yscrollcommand, yScrollCommand, ScrollCommand

WIDGET-SPECIFIC OPTIONS

-columns, columns, Columns
-displaycolumns, displayColumns, DisplayColumns
-height, height, Height
-padding, padding, Padding
-selectmode, selectMode, SelectMode
-show, show, Show

    tree

    headings

WIDGET COMMAND

pathname bbox item ?column?  
pathname cget option  
pathname children item ?newchildren?  
pathname column column ?-option ?value -option value...?  
    -id name  
    -anchor  
    -minwidth  
    -stretch  
    -width w  

pathname configure ?option? ?value option value ...?
pathname **delete** itemList
pathname **detach** itemList
pathname **exists** item
pathname **focus** ?item?
pathname **heading** column ?-option ?value -option value...
  -text text
  -image imageName
  -anchor anchor
  -command script
pathname **identify** component x y
  pathname **identify** row x y
  pathname **identify** column x y
pathname **index** item
pathname **insert** parent index ?-id id? options...
pathname **instate** statespec ?script?
pathname **item** item ?-option ?value -option value...
pathname **move** item parent index
pathname **next** item
pathname **parent** item
pathname **prev** item
pathname **see** item
pathname **selection** ?selop itemList?
  pathname **selection** set itemList
  pathname **selection** add itemList
  pathname **selection** remove itemList
  pathname **selection** toggle itemList
pathname **set** item ?column? ?value?
pathname **state** ?stateSpec?
pathname **tag** args...
  pathname **tag** bind tagName ?sequence? ?script?
  pathname **tag** configure tagName ?option? ?value option value...?
pathname **xview** args
pathname **yview** args

ITEM OPTIONS
  -text, text, Text
  -image, image, Image
NAME

ttk::treeview - hierarchical multicolumn data display widget

SYNOPSIS

 ttk::treeview pathname ?options?

DESCRIPTION

The ttk::treeview widget displays a hierarchical collection of items. Each item has a textual label, an optional image, and an optional list of data values. The data values are displayed in successive columns after the tree label.

The order in which data values are displayed may be controlled by setting the -displaycolumns widget option. The tree widget can also display column headings. Columns may be accessed by number or by symbolic names listed in the -columns widget option; see COLUMN IDENTIFIERS.

Each item is identified by a unique name. The widget will generate item IDs if they are not supplied by the caller. There is a distinguished root
item, named {}. The root item itself is not displayed; its children appear at the top level of the hierarchy.

Each item also has a list of tags, which can be used to associate event bindings with individual items and control the appearance of the item.

Treeview widgets support horizontal and vertical scrolling with the standard [-xy]scrollcommand options and [xy]view widget commands.

<table>
<thead>
<tr>
<th>STANDARD OPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>-class</td>
</tr>
<tr>
<td>-cursor, cursor, Cursor</td>
</tr>
<tr>
<td>-style</td>
</tr>
<tr>
<td>-takefocus, takeFocus, TakeFocus</td>
</tr>
<tr>
<td>-xscrollcommand, xScrollCommand, ScrollCommand</td>
</tr>
<tr>
<td>-yscrollcommand, yScrollCommand, ScrollCommand</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WIDGET-SPECIFIC OPTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-Line Name: -columns</td>
</tr>
<tr>
<td>Database Name: columns</td>
</tr>
<tr>
<td>Database Class: Columns</td>
</tr>
<tr>
<td>A list of column identifiers, specifying the number of columns and their names.</td>
</tr>
</tbody>
</table>

| Command-Line Name: -displaycolumns |
| Database Name: displayColumns |
| Database Class: DisplayColumns |
| A list of column identifiers (either symbolic names or integer indices) specifying which data columns are displayed and the order in which they appear, or the string #all. |

If set to #all (the default), all columns are shown in the order given.

| Command-Line Name: -height |
| Database Name: height |
| Database Class: Height |
Specifies the number of rows which should be visible. Note: the requested width is determined from the sum of the column widths.

Command-Line Name: -padding
Database Name: padding
Database Class: Padding
  Specifies the internal padding for the widget. The padding is a list of up to four length specifications; see Ttk::GetPaddingFromObj() for details.

Command-Line Name: -selectmode
Database Name: selectMode
Database Class: SelectMode
  Controls how the built-in class bindings manage the selection. One of extended, browse, or none.

  If set to extended (the default), multiple items may be selected. If browse, only a single item will be selected at a time. If none, the selection will not be changed.

  Note that application code and tag bindings can set the selection however they wish, regardless of the value of -selectmode.

Command-Line Name: -show
Database Name: show
Database Class: Show
  A list containing zero or more of the following values, specifying which elements of the tree to display.

  tree
    Display tree labels in column #0.

  headings
    Display the heading row.

  The default is tree headings, i.e., show all elements.

  NOTE: Column #0 always refers to the tree column, even if -show tree is not specified.
**WIDGET COMMAND**

`pathname bbox item ?column?`  
Returns the bounding box (relative to the treeview widget's window) of the specified `item` in the form `x y width height`. If `column` is specified, returns the bounding box of that cell. If the `item` is not visible (i.e., if it is a descendant of a closed item or is scrolled offscreen), returns the empty list.

`pathname cget option`  
Returns the current value of the specified `option`; see `ttk::widget(n)`.

`pathname children item ?newchildren?`  
If `newchildren` is not specified, returns the list of children belonging to `item`.

If `newchildren` is specified, replaces `item`'s child list with `newchildren`. Items in the old child list not present in the new child list are detached from the tree. None of the items in `newchildren` may be an ancestor of `item`.

`pathname column column ?-option ?value -option value...?`  
Query or modify the options for the specified `column`. If no `-option` is specified, returns a dictionary of option/value pairs. If a single `-option` is specified, returns the value of that option. Otherwise, the options are updated with the specified values. The following options may be set on each column:

- **-id name**  
The column name. This is a read-only option. For example, `[$pathname column #n -id]` returns the data column associated with display column `#n`.

- **-anchor**  
Specifies how the text in this column should be aligned with respect to the cell. One of `n, ne, e, se, s, sw, w, nw, or center`. 
-minwidth
The minimum width of the column in pixels. The treeview widget will not make the column any smaller than -minwidth when the widget is resized or the user drags a column separator.

-stretch
Specifies whether or not the column's width should be adjusted when the widget is resized.

-width w
The width of the column in pixels. Default is something reasonable, probably 200 or so.

Use pathname column #0 to configure the tree column.

pathname configure ?option? ?value option value ...?
Modify or query widget options; see ttk::widget(n).

pathname delete itemList
Deletes each of the items in itemList and all of their descendants. The root item may not be deleted. See also: detach.

pathname detach itemList
Unlinks all of the specified items in itemList from the tree. The items and all of their descendants are still present and may be reinserted at another point in the tree but will not be displayed. The root item may not be detached. See also: delete.

pathname exists item
Returns 1 if the specified item is present in the tree, 0 otherwise.

pathname focus ?item?
If item is specified, sets the focus item to item. Otherwise, returns the current focus item, or {} if there is none.

pathname heading column ?-option ?value -option value...?
Query or modify the heading options for the specified column. Valid options are:
-text text
   The text to display in the column heading.

-image imageName
   Specifies an image to display to the right of the column heading.

-anchor anchor
   Specifies how the heading text should be aligned. One of the standard Tk anchor values.

-command script
   A script to evaluate when the heading label is pressed.

Use pathname heading #0 to configure the tree column heading.

pathname identify component x y
   Returns a description of the specified component under the point given by x and y, or the empty string if no such component is present at that position. The following subcommands are supported:

pathname identify row x y
   Returns the item ID of the item at position y.

pathname identify column x y
   Returns the data column identifier of the cell at position x. The tree column has ID #0.

See COLUMN IDENTIFIERS for a discussion of display columns and data columns.

pathname index item
   Returns the integer index of item within its parent's list of children.

pathname insert parent index ?-id id? options...
   Creates a new item. parent is the item ID of the parent item, or the empty string {} to create a new top-level item. index is an integer,
or the value **end**, specifying where in the list of `parent`'s children to insert the new item. If `index` is less than or equal to zero, the new node is inserted at the beginning; if `index` is greater than or equal to the current number of children, it is inserted at the end. If `-id` is specified, it is used as the item identifier; `id` must not already exist in the tree. Otherwise, a new unique identifier is generated.

`pathname insert` returns the item identifier of the newly created item. See **ITEM OPTIONS** for the list of available options.

`pathname instate statespec ?script?`
Test the widget state; see `ttk::widget(n)`.

`pathname item item ?-option ?value -option value...?`
Query or modify the options for the specified `item`. If no `-option` is specified, returns a dictionary of option/value pairs. If a single `-option` is specified, returns the value of that option. Otherwise, the item's options are updated with the specified values. See **ITEM OPTIONS** for the list of available options.

`pathname move item parent index`
Moves `item` to position `index` in `parent`'s list of children. It is illegal to move an item under one of its descendants.

If `index` is less than or equal to zero, `item` is moved to the beginning; if greater than or equal to the number of children, it is moved to the end.

`pathname next item`
Returns the identifier of `item`'s next sibling, or `{}` if `item` is the last child of its parent.

`pathname parent item`
Returns the ID of the parent of `item`, or `{}` if `item` is at the top level of the hierarchy.

`pathname prev item`
Returns the identifier of `item`'s previous sibling, or `{}` if `item` is the
first child of its parent.

pathname see item
Ensure that item is visible: sets all of item's ancestors to -open true, and scrolls the widget if necessary so that item is within the visible portion of the tree.

pathname selection ?selop itemList?
If selop is not specified, returns the list of selected items. Otherwise, selop is one of the following:

pathname selection set itemList
itemList becomes the new selection.

pathname selection add itemList
Add itemList to the selection

pathname selection remove itemList
Remove itemList from the selection

pathname selection toggle itemList
Toggle the selection state of each item in itemList.

pathname set item ?column? ?value?
With one argument, returns a dictionary of column/value pairs for the specified item. With two arguments, returns the current value of the specified column. With three arguments, sets the value of column column in item item to the specified value. See also COLUMN IDENTIFIERS.

pathname state ?stateSpec?
Modify or query the widget state; see ttk::widget(n).

pathName tag args...

pathName tag bind tagName ?sequence? ?script?
Add a Tk binding script for the event sequence sequence to the tag tagName. When an X event is delivered to an item, binding scripts for each of the item's -tags are evaluated in
order as per `bindtags(n)`.

`<KeyPress>`, `<KeyRelease>`, and virtual events are sent to the focus item. `<ButtonPress>`, `<ButtonRelease>`, and `<Motion>` events are sent to the item under the mouse pointer. No other event types are supported.

The binding script undergoes %-substitutions before evaluation; see `bind(n)` for details.

`pathName tag configure tagName ?option? ?value option value...?`

Query or modify the options for the specified `tagName`. If one or more `option/value` pairs are specified, sets the value of those options for the specified tag. If a single `option` is specified, returns the value of that option (or the empty string if the option has not been specified for `tagName`). With no additional arguments, returns a dictionary of the option settings for `tagName`. See `TAG OPTIONS` for the list of available options.

`pathName xview args`

Standard command for horizontal scrolling; see `widget(n)`.

`pathName yview args`

Standard command for vertical scrolling; see `ttk::widget(n)`.

---

**ITEM OPTIONS**

The following item options may be specified for items in the `insert` and `item` widget commands.

Command-Line Name: `-text`
Database Name: `text`
Database Class: `Text`

The textual label to display for the item.

Command-Line Name: `-image`
Database Name: **image**
Database Class: **Image**

A Tk image, displayed to the left of the label.

Command-Line Name: **-values**
Database Name: **values**
Database Class: **Values**

The list of values associated with the item.

Each item should have the same number of values as the `-columns` widget option. If there are fewer values than columns, the remaining values are assumed empty. If there are more values than columns, the extra values are ignored.

Command-Line Name: **-open**
Database Name: **open**
Database Class: **Open**

A boolean value indicating whether the item's children should be displayed (`-open true`) or hidden (`-open false`).

Command-Line Name: **-tags**
Database Name: **tags**
Database Class: **Tags**

A list of tags associated with this item.

**TAG OPTIONS**

The following options may be specified on tags:

- **foreground**
  Specifies the text foreground color.

- **background**
  Specifies the cell or item background color.

- **font**
  Specifies the font to use when drawing text.

- **image**
Specifies the item image, in case the item's -image option is empty.

(@@@@ TODO: sort out order of precedence for options)

COLUMN IDENTIFIERS

Column identifiers take any of the following forms:

- A symbolic name from the list of -columns.
- An integer $n$, specifying the $n$th data column.
- A string of the form #n, where $n$ is an integer, specifying the $n$th display column.

NOTE: Item -values may be displayed in a different order than the order in which they are stored.

NOTE: Column #0 always refers to the tree column, even if -show tree is not specified.

A data column number is an index into an item's -values list; a display column number is the column number in the tree where the values are displayed. Tree labels are displayed in column #0. If -displaycolumns is not set, then data column $n$ is displayed in display column #n+1. Again, column #0 always refers to the tree column.

VIRTUAL EVENTS

The treeview widget generates the following virtual events.

<<TreeviewSelect>>
   Generated whenever the selection changes.

<<TreeviewOpen>>
   Generated just before setting the focus item to -open true.

<<TreeviewClose>>
   Generated just after setting the focus item to -open false.
The **focus** and **selection** widget commands can be used to determine the affected item or items.

**SEE ALSO**

 ttk::widget, listbox, image, bind

Copyright © 1995-1997 Roger E. Critchlow Jr.  
Copyright © 2004 Joe English
NAME
console - Control the console on systems without a real console

SYNOPSIS

DESCRIPTION

console eval script
console hide
console show
console title ?string?

ACCESS TO THE MAIN INTERPRETER

consoleinterp eval script
consoleinterp record script

ADDITIONAL TRAP CALLS

DEFAULT BINDINGS

EXAMPLE

SEE ALSO

KEYWORDS

NAME

console - Control the console on systems without a real console

SYNOPSIS

console subcommand ?arg ...?

DESCRIPTION

The console window is a replacement for a real console to allow input and output on the standard I/O channels on platforms that do not have a real console. It is implemented as a separate interpreter with the Tk toolkit loaded, and control over this interpreter is given through the
**console** command. The behaviour of the console window is defined mainly through the contents of the **console.tcl** file in the Tk library. Except for TkAqua, this command is not available when Tk is loaded into a tclsh interpreter with “**package require Tk**”, as a conventional terminal is expected to be present in that case. In TkAqua, this command is only available when stdin is **/dev/null** (as is the case e.g. when the application embedding Tk is started from the Mac OS X Finder).

**console eval** *script*

Evaluate the *script* argument as a Tcl script in the console interpreter. The normal interpreter is accessed through the **consoleinterp** command in the console interpreter.

**console hide**

Hide the console window from view. Precisely equivalent to withdrawing the . window in the console interpreter.

**console show**

Display the console window. Precisely equivalent to deiconifying the . window in the console interpreter.

**console title** *string*?

Query or modify the title of the console window. If *string* is not specified, queries the title of the console window, and sets the title of the console window to *string* otherwise. Precisely equivalent to using the **wm title** command in the console interpreter.

### ACCESS TO THE MAIN INTERPRETER

The **consoleinterp** command in the console interpreter allows scripts to be evaluated in the main interpreter. It supports two subcommands: **eval** and **record**.

**consoleinterp eval** *script*

Evaluates *script* as a Tcl script at the global level in the main interpreter.
**console interp record** *script*

Records and evaluates *script* as a Tcl script at the global level in the main interpreter as if *script* had been typed in at the console.

### ADDITIONAL TRAP CALLS

There are several additional commands in the console interpreter that are called in response to activity in the main interpreter. *These are documented here for completeness only; they form part of the internal implementation of the console and are likely to change or be modified without warning.*

Output to the console from the main interpreter via the stdout and stderr channels is handled by invoking the `tk::ConsoleOutput` command in the console interpreter with two arguments. The first argument is the name of the channel being written to, and the second argument is the string being written to the channel (after encoding and end-of-line translation processing has been performed.)

When the . window of the main interpreter is destroyed, the `tk::ConsoleExit` command in the console interpreter is called (assuming the console interpreter has not already been deleted itself, that is.)

### DEFAULT BINDINGS

The default script creates a console window (implemented using a text widget) that has the following behaviour:

[1]

Pressing the tab key inserts a TAB character (as defined by the Tcl `\t` escape.)

[2]

Pressing the return key causes the current line (if complete by the rules of `info complete`) to be passed to the main interpreter for evaluation.
Pressing the delete key deletes the selected text (if any text is selected) or the character to the right of the cursor (if not at the end of the line.)

Pressing the backspace key deletes the selected text (if any text is selected) or the character to the left of the cursor (of not at the start of the line.)

Pressing either Control+A or the home key causes the cursor to go to the start of the line (but after the prompt, if a prompt is present on the line.)

Pressing either Control+E or the end key causes the cursor to go to the end of the line.

Pressing either Control+P or the up key causes the previous entry in the command history to be selected.

Pressing either Control+N or the down key causes the next entry in the command history to be selected.

Pressing either Control+B or the left key causes the cursor to move one character backward as long as the cursor is not at the prompt.

Pressing either Control+F or the right key causes the cursor to move one character forward.

Pressing F9 rebuilds the console window by destroying all its children and reloading the Tcl script that defined the console's
behaviour.

Most other behaviour is the same as a conventional text widget except for the way that the \texttt{\textlangle\textrangle\textrangle} event is handled identically to the \texttt{\textlangle\textrangle\textrangle} event.

\section*{EXAMPLE}

Not all platforms have the \texttt{console} command, so debugging code often has the following code fragment in it so output produced by \texttt{puts} can be seen while during development:

\begin{verbatim}
  catch {console show}
\end{verbatim}

\section*{SEE ALSO}

\texttt{destroy}, \texttt{fconfigure}, \texttt{history}, \texttt{interp}, \texttt{puts}, \texttt{text}, \texttt{wm}

\section*{KEYWORDS}

\texttt{console}, \texttt{interpreter}, \texttt{window}, \texttt{interactive}, \texttt{output channels}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2001 Donal K. Fellows
loadTk - Load Tk into a safe interpreter.

::safe::loadTk slave ?-use windowId? ?-display displayName?

Safe Tk is based on Safe Tcl, which provides a mechanism that allows restricted and mediated access to auto-loading and packages for safe interpreters. Safe Tk adds the ability to configure the interpreter for safe Tk operations and load Tk into safe interpreters.

The ::safe::loadTk command initializes the required data structures in the named safe interpreter and then loads Tk into it. The interpreter must have been created with ::safe::interpCreate or have been initialized with ::safe::interpInit. The command returns the name of the safe interpreter. If -use is specified, the window identified by the specified system dependent identifier windowId is used to contain the “.” window of the safe interpreter; it can be any valid id, eventually referencing a window belonging to another application. As a convenience, if the window you plan to use is a Tk Window of the application you can use the window name (e.g. .x.y) instead of its window Id ([winfo id .x.y]). When -use is not specified, a new toplevel window is created for the “.” window of the safe interpreter. On X11 if you want the embedded window to use another display than the default one, specify it with -display. See the SECURITY ISSUES section below for implementation details.
Please read the **safe** manual page for Tcl to learn about the basic security considerations for Safe Tcl.

::safe::loadTk adds the value of **tk_library** taken from the master interpreter to the virtual access path of the safe interpreter so that auto-loading will work in the safe interpreter.

Tk initialization is now safe with respect to not trusting the slave's state for startup. ::safe::loadTk registers the slave's name so when the Tk initialization (**Tk_Safelnit**) is called and in turn calls the master's ::safe::InitTk it will return the desired **argv** equivalent (**-use windowId**, correct **-display**, etc.)

When **-use** is not used, the new toplevel created is specially decorated so the user is always aware that the user interface presented comes from a potentially unsafe code and can easily delete the corresponding interpreter.

On X11, conflicting **-use** and **-display** are likely to generate a fatal X error.

---

**SEE ALSO**

**safe**, **interp**, **library**, **load**, **package**, **source**, **unknown**

**KEYWORDS**


---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME

selection - Manipulate the X selection

SYNOPSIS

DESCRIPTION

This command provides a Tcl interface to the X selection mechanism and implements the full selection functionality described in the X Inter-Client Communication Conventions Manual (ICCCM).

Note that for management of the CLIPBOARD selection (see below), the clipboard command may also be used.
The first argument to `selection` determines the format of the rest of the arguments and the behavior of the command. The following forms are currently supported:

**selection clear ?-displayof window? ?-selection selection?**

If `selection` exists anywhere on `window`'s display, clear it so that no window owns the selection anymore. `Selection` specifies the X selection that should be cleared, and should be an atom name such as PRIMARY or CLIPBOARD; see the Inter-Client Communication Conventions Manual for complete details. `Selection` defaults to PRIMARY and `window` defaults to “.”. Returns an empty string.

**selection get ?-displayof window? ?-selection selection? ?-type type?**

Retrieves the value of `selection` from `window`'s display and returns it as a result. `Selection` defaults to PRIMARY and `window` defaults to “.”. `Type` specifies the form in which the selection is to be returned (the desired “target” for conversion, in ICCCM terminology), and should be an atom name such as STRING or FILE_NAME; see the Inter-Client Communication Conventions Manual for complete details. `Type` defaults to STRING. The selection owner may choose to return the selection in any of several different representation formats, such as STRING, UTF8_STRING, ATOM, INTEGER, etc. (this format is different than the selection type; see the ICCCM for all the confusing details). If the selection is returned in a non-string format, such as INTEGER or ATOM, the `selection` command converts it to string format as a collection of fields separated by spaces: atoms are converted to their textual names, and anything else is converted to hexadecimal integers. Note that `selection get` does not retrieve the selection in the UTF8_STRING format unless told to.

**selection handle ?-selection s? ?-type t? ?-format f? window command**

Creates a handler for selection requests, such that `command` will be executed whenever selection `s` is owned by `window` and someone attempts to retrieve it in the form given by type `t` (e.g. `t` is
specified in the **selection get** command). S defaults to PRIMARY, t
defaults to STRING, and f defaults to STRING. If **command** is an
empty string then any existing handler for **window**, t, and s is
removed. Note that when the selection is handled as type STRING
it is also automatically handled as type UTF8_STRING as well.

When **selection** is requested, **window** is the selection owner, and
**type** is the requested type, **command** will be executed as a Tcl
command with two additional numbers appended to it (with space
separators). The two additional numbers are **offset** and **maxChars**:  
**offset** specifies a starting character position in the selection and
**maxChars** gives the maximum number of characters to retrieve.
The command should return a value consisting of at most
**maxChars** of the selection, starting at position **offset**. For very large
selections (larger than **maxChars**) the selection will be retrieved
using several invocations of **command** with increasing **offset**
values. If **command** returns a string whose length is less than
**maxChars**, the return value is assumed to include all of the
remainder of the selection; if the length of **command**'s result is
equal to **maxChars** then **command** will be invoked again, until it
eventually returns a result shorter than **maxChars**. The value of
**maxChars** will always be relatively large (thousands of characters).

If **command** returns an error then the selection retrieval is rejected
just as if the selection did not exist at all.

The **format** argument specifies the representation that should be
used to transmit the selection to the requester (the second column
of Table 2 of the ICCCM), and defaults to STRING. If **format** is
STRING, the selection is transmitted as 8-bit ASCII characters (i.e.
just in the form returned by **command**, in the system **encoding**; the
UTF8_STRING format always uses UTF-8 as its encoding). If
**format** is ATOM, then the return value from **command** is divided into
fields separated by white space; each field is converted to its atom
value, and the 32-bit atom value is transmitted instead of the atom
name. For any other **format**, the return value from **command** is
divided into fields separated by white space and each field is
converted to a 32-bit integer; an array of integers is transmitted to
the selection requester.

The *format* argument is needed only for compatibility with selection requesters that do not use Tk. If Tk is being used to retrieve the selection then the value is converted back to a string at the requesting end, so *format* is irrelevant.

**selection own** `-displayof window?` `-selection selection?`

**selection own** `-command command?` `-selection selection?` *window*

The first form of **selection own** returns the path name of the window in this application that owns *selection* on the display containing *window*, or an empty string if no window in this application owns the selection. *Selection* defaults to PRIMARY and *window* defaults to “.”.

The second form of **selection own** causes *window* to become the new owner of *selection* on *window*’s display, returning an empty string as result. The existing owner, if any, is notified that it has lost the selection. If *command* is specified, it is a Tcl script to execute when some other window claims ownership of the selection away from *window*. *Selection* defaults to PRIMARY.

**EXAMPLES**

On X11 platforms, one of the standard selections available is the SECONDARY selection. Hardly anything uses it, but here is how to read it using Tk:

```tcl
set selContents [selection get -selection SECONDARY]
```

Many different types of data may be available for a selection; the special type TARGETS allows you to get a list of available types:

```tcl
foreach type [selection get -type TARGETS] {
```
puts "Selection PRIMARY supports type $type"
}

To claim the selection, you must first set up a handler to supply the data for the selection. Then you have to claim the selection...

```tcl
# Set up the data handler ready for incoming request
set foo "This is a string with some data in it... blah blah"
selection handle -selection SECONDARY . getData
proc getData {offset maxChars} {
    puts "Retrieving selection starting at $offset"
    return [string range $::foo $offset [expr {$offset+$maxChars}]]
}

# Now we grab the selection itself
puts "Claiming selection"
selection own -command lost -selection SECONDARY .
proc lost {} {
    puts "Lost selection"
}
```

SEE ALSO

clipboard

KEYWORDS

clear, format, handler, ICCCM, own, selection, target, type
NAME

tk_messageBox - pops up a message window and waits for user response.

SYNOPSIS

DESCRIPTION

-default name
-detail string
-icon iconImage
-message string
-parent window
-title titleString
-type predefinedType
  abortretryignore
  ok
  okcancel
  retrycancel
  yesno
  yesnocancel

EXAMPLE

KEYWORDS
This procedure creates and displays a message window with an application-specified message, an icon and a set of buttons. Each of the buttons in the message window is identified by a unique symbolic name (see the -type options). After the message window is popped up, tk_messageBox waits for the user to select one of the buttons. Then it returns the symbolic name of the selected button. The following option-value pairs are supported:

-default name
Name gives the symbolic name of the default button for this message window ("ok", "cancel", and so on). See -type for a list of the symbolic names. If this option is not specified, the first button in the dialog will be made the default.

-detail string
Specifies an auxiliary message to the main message given by the -message option. Where supported by the underlying OS, the message detail will be presented in a less emphasized font than the main message.

-icon iconImage
Specifies an icon to display. IconImage must be one of the following: error, info, question or warning. If this option is not specified, then the info icon will be displayed.

-message string
Specifies the message to display in this message box.

-parent window
Makes window the logical parent of the message box. The message box is displayed on top of its parent window.

-title titleString
Specifies a string to display as the title of the message box. The default value is an empty string.

-type predefinedType
Arranges for a predefined set of buttons to be displayed. The
following values are possible for `predefinedType`:

**abortretryignore**
Displays three buttons whose symbolic names are **abort**, **retry** and **ignore**.

**ok**
Displays one button whose symbolic name is **ok**.

**okcancel**
Displays two buttons whose symbolic names are **ok** and **cancel**.

**retrycancel**
Displays two buttons whose symbolic names are **retry** and **cancel**.

**yesno**
Displays two buttons whose symbolic names are **yes** and **no**.

**yesnocancel**
Displays three buttons whose symbolic names are **yes**, **no** and **cancel**.

---

**EXAMPLE**

```bash
set answer [tk_messageBox -message "Really quit?" \\
    -icon question -type yesno \\
    -detail "Select "Yes" to make the application exit"]
switch -- $answer {
    yes exit
    no {tk_messageBox -message "I know you like this" 
        -type ok}
}
```

---

**KEYWORDS**
message box

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
NAME
   ttk::frame - Simple container widget

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS
   -class
   -cursor, cursor, Cursor
   -style
   -takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS
   -borderwidth, borderWidth, BorderWidth
   -relief, relief, Relief
   -padding, padding, Padding
   -width, width, Width
   -height, height, Height

NOTES

SEE ALSO

KEYWORDS

NAME

   ttk::frame - Simple container widget

SYNOPSIS

   ttk::frame pathName ?options?

DESCRIPTION

   A ttk::frame widget is a container, used to group other widgets together.
STANDARD OPTIONS

-class
-cursor, cursor, Cursor
-style
-takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS

Command-Line Name: -borderwidth
Database Name: borderWidth
Database Class: BorderWidth
   The desired width of the widget border. Defaults to 0.

Command-Line Name: -relief
Database Name: relief
Database Class: Relief
   One of the standard Tk border styles: flat, groove, raised, ridge, solid, or sunken. Defaults to flat.

Command-Line Name: -padding
Database Name: padding
Database Class: Padding
   Additional padding to include inside the border.

Command-Line Name: -width
Database Name: width
Database Class: Width
   If specified, the widget's requested width in pixels.

Command-Line Name: -height
Database Name: height
Database Class: Height
   If specified, the widget's requested height in pixels.

WIDGET COMMAND

Supports the standard widget commands configure, cget, identify,
instate, and state; see ttk::widget(n).

NOTES

Note that if the pack, grid, or other geometry managers are used to manage the children of the frame, by the GM's requested size will normally take precedence over the frame widget's -width and -height options. pack propagate and grid propagate can be used to change this.

SEE ALSO

ttk::widget, ttk::labelframe, frame

KEYWORDS

widget, frame, container

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2005 Joe English
NAME
ttk::widget - Standard options and commands supported by Tk themed widgets

DESCRIPTION

STANDARD OPTIONS
-class, undefined, undefined
-cursor, cursor, Cursor
-takefocus, takeFocus, TakeFocus
-style, style, Style

SCROLLABLE WIDGET OPTIONS
-xscrollcommand, xScrollCommand, ScrollCommand
-yscrollcommand, yScrollCommand, ScrollCommand

LABEL OPTIONS
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-image, image, Image
-compound, compound, Compound
  text
  image
  center
  top
  bottom
  left
  right
  none
-width, width, Width

COMPATIBILITY OPTIONS
-state, state, State

COMMANDS
\texttt{\textit{pathName} \texttt{\textit{cget} \textit{option}}}
tkk::widget - Standard options and commands supported by Tk themed widgets

DESCRIPTION

This manual describes common widget options and commands.

STANDARD OPTIONS

The following options are supported by all Tk themed widgets:

Command-Line Name: -class
Database Name: undefined
Database Class: undefined

Specifies the window class. The class is used when querying the option database for the window's other options, to determine the default bindtags for the window, and to select the widget's default
layout and style. This is a read-only option: it may only be specified when the window is created, and may not be changed with the `configure` widget command.

Command-Line Name: **-cursor**
Database Name: **cursor**
Database Class: **Cursor**
   Specifies the mouse cursor to be used for the widget. See `Tk_GetCursor` and `cursors(n)` in the Tk reference manual for the legal values. If set to the empty string (the default), the cursor is inherited from the parent widget.

Command-Line Name: **-takefocus**
Database Name: **takeFocus**
Database Class: **TakeFocus**
   Determines whether the window accepts the focus during keyboard traversal. Either 0, 1, a command prefix (to which the widget path is appended, and which should return 0 or 1), or the empty string. See `options(n)` in the Tk reference manual for the full description.

Command-Line Name: **-style**
Database Name: **style**
Database Class: **Style**
   May be used to specify a custom widget style.

**SCROLLABLE WIDGET OPTIONS**

The following options are supported by widgets that are controllable by a scrollbar. See `scrollbar(n)` for more information

Command-Line Name: **-xscrollcommand**
Database Name: **xScrollCommand**
Database Class: **ScrollCommand**
   A command prefix, used to communicate with horizontal scrollbars. When the view in the widget's window changes, the widget will generate a Tcl command by concatenating the scroll command and two numbers. Each of the numbers is a fraction between 0 and 1 indicating a position in the document; 0 indicates the beginning,
and 1 indicates the end. The first fraction indicates the first information in the widget that is visible in the window, and the second fraction indicates the information just after the last portion that is visible.

Typically the \texttt{xScrollCommand} option consists of the path name of a \texttt{scrollbar} widget followed by “set”, e.g. “.x.scrollbar set”. This will cause the scrollbar to be updated whenever the view in the window changes.

If this option is set to the empty string (the default), then no command will be executed.

Command-Line Name: \texttt{-yscrollcommand}
Database Name: \texttt{yScrollCommand}
Database Class: \texttt{ScrollCommand}
A command prefix, used to communicate with vertical scrollbars.
See the description of \texttt{-xscrollcommand} above for details.

\section*{LABEL OPTIONS}

The following options are supported by labels, buttons, and other button-like widgets:

Command-Line Name: \texttt{-text}
Database Name: \texttt{text}
Database Class: \texttt{Text}
Specifies a text string to be displayed inside the widget (unless overridden by \texttt{-textvariable}).

Command-Line Name: \texttt{-textvariable}
Database Name: \texttt{textVariable}
Database Class: \texttt{Variable}
Specifies the name of variable whose value will be used in place of the \texttt{-text} resource.

Command-Line Name: \texttt{-underline}
Database Name: \texttt{underline}
Database Class: **Underline**
If set, specifies the integer index (0-based) of a character to underline in the text string. The underlined character is used for mnemonic activation.

Command-Line Name: **-image**
Database Name: **image**
Database Class: **Image**
Specifies an image to display. This is a list of 1 or more elements. The first element is the default image name. The rest of the list is a sequence of statespec / value pairs as per **style map**, specifying different images to use when the widget is in a particular state or combination of states. All images in the list should have the same size.

Command-Line Name: **-compound**
Database Name: **compound**
Database Class: **Compound**
Specifies how to display the image relative to the text, in the case both **-text** and **-image** are present. Valid values are:

- **text**
  Display text only.

- **image**
  Display image only.

- **center**
  Display text centered on top of image.

- **top**

- **bottom**

- **left**

- **right**
  Display image above, below, left of, or right of the text, respectively.
none
The default; display the image if present, otherwise the text.

Command-Line Name: **-width**
Database Name: **width**
Database Class: **Width**
If greater than zero, specifies how much space, in character widths, to allocate for the text label. If less than zero, specifies a minimum width. If zero or unspecified, the natural width of the text label is used.

**COMPATIBILITY OPTIONS**

Command-Line Name: **-state**
Database Name: **state**
Database Class: **State**
May be set to **normal** or **disabled** to control the **disabled** state bit. This is a write-only option: setting it changes the widget state, but the **state** widget command does not affect the **-state** option.

**COMMANDS**

*pathName cget option*
Returns the current value of the configuration option given by **option**.

*pathName configure ?option? ?value option value ...?*
Query or modify the configuration options of the widget. If one or more **option-value** pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. If **option** is specified with no **value**, then the command returns a list describing the named option: the elements of the list are the option name, database name, database class, default value, and current value. If no **option** is specified, returns a list describing all of the available options for **pathName**.
\textit{pathName identify} \texttt{x y}

Returns the name of the element under the point given by \texttt{x} and \texttt{y},
or an empty string if the point does not lie within any element. \texttt{x} and \texttt{y}
are pixel coordinates relative to the widget.

\textit{pathName instate} \texttt{statespec} ?\texttt{script}?

Test the widget's state. If \texttt{script} is not specified, returns 1 if the
widget state matches \texttt{statespec} and 0 otherwise. If \texttt{script} is
specified, equivalent to

\begin{center}
\begin{verbatim}
if {[\textit{pathName instate} \texttt{stateSpec}]} \texttt{script}
\end{verbatim}
\end{center}

\textit{pathName state} ?\texttt{stateSpec}?

Modify or inquire widget state. If \texttt{stateSpec} is present, sets the
widget state: for each flag in \texttt{stateSpec}, sets the corresponding flag
or clears it if prefixed by an exclamation point.

Returns a new state spec indicating which flags were changed:

\begin{center}
\begin{verbatim}
set changes [\textit{pathName state spec}]
\end{verbatim}
\end{center}

\begin{verbatim}
\textit{pathName state} $changes
\end{verbatim}

will restore \textit{pathName} to the original state. If \texttt{stateSpec} is not
specified, returns a list of the currently-enabled state flags.

\textbf{WIDGET STATES}

The widget state is a bitmap of independent state flags. Widget state
flags include:

\textbf{active}

The mouse cursor is over the widget and pressing a mouse button
will cause some action to occur. (aka “prelight” (Gnome), “hot”
(Windows), “hover”).

\textbf{disabled}
Widget is disabled under program control (aka “unavailable”, “inactive”)

**focus**
Widget has keyboard focus

**pressed**
Widget is being pressed (aka “armed” in Motif).

**selected**
“On”, “true”, or “current” for things like checkbuttons and radiobuttons.

**background**
Windows and the Mac have a notion of an “active” or foreground window. The **background** state is set for widgets in a background window, and cleared for those in the foreground window.

**readonly**
Widget should not allow user modification.

**alternate**
A widget-specific alternate display format. For example, used for checkbuttons and radiobuttons in the “tristate” or “mixed” state, and for buttons with **default active**.

**invalid**
The widget’s value is invalid. (Potential uses: scale widget value out of bounds, entry widget value failed validation.)

**hover**
The mouse cursor is within the widget. This is similar to the **active** state; it is used in some themes for widgets that provide distinct visual feedback for the active widget in addition to the active element within the widget.

A **state specification** or **stateSpec** is a list of state names, optionally prefixed with an exclamation point (!) indicating that the bit is off.
EXAMPLES

set b [ttk::button .b]

# Disable the widget:
$b state disabled

# Invoke the widget only if it is currently pressed
$b instate {pressed !disabled} { .b invoke }

# Reenable widget:
$b state !disabled

SEE ALSO

ttk::intro, style

KEYWORDS

state, configure, option
NAME
cursors - mouse cursors available in Tk

DESCRIPTION
The `-cursor` widget option allows a Tk programmer to change the mouse cursor for a particular widget. The cursor names recognized by Tk on all platforms are:

```
X_cursor
  arrow
  based_arrow_down
  based_arrow_up
  boat
  bogosity
  bottom_left_corner
  bottom_right_corner
  bottom_side
  bottom_tee
  box_spiral
  center_ptr
  circle
  clock
  coffee_mug
  cross
  cross_reverse
  crosshair
  diamond_cross
  dot
  dotbox
```
double_arrow
draft_large
draft_small
draped_box
exchange
fleur
gobbler
gumby
hand1
hand2
heart
icon
iron_cross
left_ptr
left_side
left_tee
leftbutton
ll_angle
lr_angle
man
middlebutton
mouse
none
pencil
pirate
plus
question_arrow
right_ptr
right_side
right_tee
rightbutton
rtl_logo
sailboat
sb_down_arrow
sb_h_double_arrow
sb_left_arrow
sb_right_arrow
The **none** cursor can be specified to eliminate the cursor.

### PORTABILITY ISSUES

**Windows**

On Windows systems, the following cursors are mapped to native cursors:

```plaintext
arrow
center_ptr
crosshair
fleur
ibeam
icon
none
```
And the following additional cursors are available:

- no
- starting
- size
- size_ne_sw
- size_ns
- size_nw_se
- size_we
- uparrow
- wait

**Mac OS X**

On Mac OS X systems, the following cursors are mapped to native cursors:

- arrow
- cross
- crosshair
- ibeam
- none
- plus
- watch
- xterm

And the following additional native cursors are available:

- copyarrow
aliasarrow
cross-hair
closedhand
openthand
pointinghand
resizeleft
resizeright
resizeleftright
resizeup
resizedown
resizeupdown
notallowed
poof
countinguphand
countingdownhand
countingupanddownhand
spinning

KEYWORDS
cursor, option

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2006-2007 Daniel A. Steffen <das(at)users.sourceforge.net>
lower - Change a window's position in the stacking order

SYNOPSIS

lower window ?belowThis?

DESCRIPTION

If the belowThis argument is omitted then the command lowers window so that it is below all of its siblings in the stacking order (it will be obscured by any siblings that overlap it and will not obscure any siblings). If belowThis is specified then it must be the path name of a window that is either a sibling of window or the descendant of a sibling of window. In this case the lower command will insert window into the stacking order just below belowThis (or the ancestor of belowThis that is a sibling of window); this could end up either raising or lowering window.

SEE ALSO

raise

KEYWORDS

lower, obscure, stacking order
NAME
send - Execute a command in a different application

SYNOPSIS

DESCRIPTION

This command arranges for cmd (and args) to be executed in the application named by app. It returns the result or error from that command execution. App may be the name of any application whose main window is on the display containing the sender's main window; it need not be within the same process. If no arg arguments are present, then the command to be executed is contained entirely within the cmd argument. If one or more args are present, they are concatenated to form the command to be executed, just as for the eval command.
If the initial arguments of the command begin with “-” they are treated as options. The following options are currently defined:

**-async**
Requests asynchronous invocation. In this case the **send** command will complete immediately without waiting for **cmd** to complete in the target application; no result will be available and errors in the sent command will be ignored. If the target application is in the same process as the sending application then the **-async** option is ignored.

**-displayof pathName**
Specifies that the target application's main window is on the display of the window given by **pathName**, instead of the display containing the application's main window.

**--**
Serves no purpose except to terminate the list of options. This option is needed only if **app** could contain a leading “-” character.

### APPLICATION NAMES

The name of an application is set initially from the name of the program or script that created the application. You can query and change the name of an application with the **tk appname** command.

### DISABLING SENDS

If the **send** command is removed from an application (e.g. with the command **rename send {}**) then the application will not respond to incoming send requests anymore, nor will it be able to issue outgoing requests. Communication can be reenabled by invoking the **tk appname** command.

### SECURITY

The **send** command is potentially a serious security loophole. On Unix, any application that can connect to your X server can send scripts to
your applications. These incoming scripts can use Tcl to read and write your files and invoke subprocesses under your name. Host-based access control such as that provided by \texttt{xhost} is particularly insecure, since it allows anyone with an account on particular hosts to connect to your server, and if disabled it allows anyone anywhere to connect to your server. In order to provide at least a small amount of security, Tk checks the access control being used by the server and rejects incoming sends unless (a) \texttt{xhost}-style access control is enabled (i.e. only certain hosts can establish connections) and (b) the list of enabled hosts is empty. This means that applications cannot connect to your server unless they use some other form of authorization such as that provide by \texttt{xauth}. Under Windows, \texttt{send} is currently disabled. Most of the functionality is provided by the \texttt{dde} command instead.

\section*{EXAMPLE}

This script fragment can be used to make an application that only runs once on a particular display.

\begin{verbatim}
if {[tk appname FoobarApp] ne "FoobarApp"} {
    send -async FoobarApp RemoteStart $argv
    exit
}
# The command that will be called remotely, which raises
# the application main window and opens the requested files
proc RemoteStart args {
    raise .
    foreach filename $args {
        OpenFile $filename
    }
}
\end{verbatim}

\section*{KEYWORDS}

\texttt{application, dde, name, remote execution, security, send}
tk_optionMenu - Create an option menubutton and its menu

SYNOPSIS

tk_optionMenu pathName varName value ?value value ...?

DESCRIPTION

This procedure creates an option menubutton whose name is pathName, plus an associated menu. Together they allow the user to select one of the values given by the value arguments. The current value will be stored in the global variable whose name is given by varName and it will also be displayed as the label in the option menubutton. The user can click on the menubutton to display a menu containing all of the values and thereby select a new value. Once a new value is selected, it will be stored in the variable and appear in the option menubutton. The current value can also be changed by setting the variable.

The return value from tk_optionMenu is the name of the menu associated with pathName, so that the caller can change its configuration options or manipulate it in other ways.

EXAMPLE

tk_optionMenu .foo myVar Foo Bar Boo Spong Wibble
pack .foo
KEYWORDS

option menu

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

ttk::intro - Introduction to the Tk theme engine

OVERVIEW

The Tk themed widget set is based on a revised and enhanced version of TIP #48 (http://tip.tcl.tk/48) specified style engine. The main concepts are described below. The basic idea is to separate, to the extent possible, the code implementing a widget's behavior from the code implementing its appearance. Widget class bindings are primarily responsible for maintaining the widget state and invoking callbacks; all aspects of the widgets appearance is

THEMES

A theme is a collection of elements and styles that determine the look and feel of the widget set. Themes can be used to:

- Isolate platform differences (X11 vs. classic Windows vs. XP vs. Aqua ...)
- Adapt to display limitations (low-color, grayscale, monochrome, tiny screens)
- Accessibility (high contrast, large type)
- Application suite branding
- Blend in with the rest of the desktop (Gnome, KDE, Java)
- And, of course: eye candy.
ELEMENTS

An element displays an individual part of a widget. For example, a vertical scrollbar widget contains uparrow, downarrow, trough and slider elements.

Element names use a recursive dotted notation. For example, uparrow identifies a generic arrow element, and Scrollbar.uparrow and Combobox.uparrow identify widget-specific elements. When looking for an element, the style engine looks for the specific name first, and if an element of that name is not found it looks for generic elements by stripping off successive leading components of the element name.

Like widgets, elements have options which specify what to display and how to display it. For example, the text element (which displays a text string) has -text, -font, -foreground, -background, -underline, and -width options. The value of an element option is taken from:

• An option of the same name and type in the widget containing the element;

• A dynamic setting specified by style map and the current state;

• The default setting specified by style configure; or

• The element's built-in default value for the option.

LAYOUTS

A layout specifies which elements make up a widget and how they are arranged. The layout engine uses a simplified version of the pack algorithm: starting with an initial cavity equal to the size of the widget, elements are allocated a parcel within the cavity along the side specified by the -side option, and placed within the parcel according to the -sticky option. For example, the layout for a horizontal scrollbar

```python
ttk::style layout Horizontal.TScrollbar {
    Scrollbar.trough -children {
```
Scrollbar.leftarrow -side left -sticky w
Scrollbar.rightarrow -side right -sticky e
Scrollbar.thumb -side left -expand true -sticky ew
}
}

By default, the layout for a widget is the same as its class name. Some widgets may override this (for example, the `ttk::scrollbar` widget chooses different layouts based on the `-orient` option).

## STATES

In standard Tk, many widgets have a `-state` option which (in most cases) is either `normal` or `disabled`. Some widgets support additional states, such as the `entry` widget which has a `readonly` state and the various flavors of buttons which have `active` state.

The themed Tk widgets generalizes this idea: every widget has a bitmap of independent state flags. Widget state flags include `active`, `disabled`, `pressed`, `focus`, etc., (see `ttk::widget(n)` for the full list of state flags).

Instead of a `-state` option, every widget now has a `state` widget command which is used to set or query the state. A `state specification` is a list of symbolic state names indicating which bits are set, each optionally prefixed with an exclamation point indicating that the bit is cleared instead.

For example, the class bindings for the `ttk::button` widget are:

```
bind TButton <Enter>    { %W state active }
binder TButton <Leave>  { %W state !active }
binder TButton <ButtonPress-1> { %W state pressed }
binder TButton <Button1-Leave> { %W state !pressed }
binder TButton <Button1-Enter> { %W state pressed }
binder TButton <ButtonRelease-1> \n```
This specifies that the widget becomes **active** when the pointer enters the widget, and inactive when it leaves. Similarly it becomes **pressed** when the mouse button is pressed, and **!pressed** on the ButtonRelease event. In addition, the button unpresses if pointer is dragged outside the widget while Button-1 is held down, and represses if it's dragged back in. Finally, when the mouse button is released, the widget's `-command` is invoked, but only if the button is currently in the **pressed** state. (The actual bindings are a little more complicated than the above, but not by much).

*Note to self: rewrite that paragraph. It's horrible.*

### STYLES

Each widget is associated with a *style*, which specifies values for element options. Style names use a recursive dotted notation like layouts and elements; by default, widgets use the class name to look up a style in the current theme. For example:

```tcl
ttk::style configure TButton \
   -background #d9d9d9 \n   -foreground black \n   -relief raised \
```

Many elements are displayed differently depending on the widget state. For example, buttons have a different background when they are active, a different foreground when disabled, and a different relief when pressed. The *style map* command specifies dynamic option settings for a particular style:

```tcl
ttk::style map TButton \
```
SEE ALSO

ttk::widget, ttk::style
NAME

ttk_image - Define an element based on an image

SYNOPSIS

DESCRIPTION

OPTIONS

-border padding
-height height
-padding padding
-sticky spec
-width width

IMAGE STRETCHING

EXAMPLE

SEE ALSO

KEYWORDS

NAME

ttk_image - Define an element based on an image

SYNOPSIS

**ttk::style element create** name **image imageSpec** ?options?

DESCRIPTION

The *image* element factory creates a new element in the current theme whose visual appearance is determined by Tk images. *imageSpec* is a list of one or more elements. The first element is the default image name. The rest of the list is a sequence of *statespec / value* pairs specifying other images to use when the element is in a particular state or combination of states.
Valid options are:

**-border** *padding*
*padding* is a list of up to four integers, specifying the left, top, right, and bottom borders, respectively. See **IMAGE STRETCHING**, below.

**-height** *height*
Specifies a minimum height for the element. If less than zero, the base image's height is used as a default.

**-padding** *padding*
Specifies the element's interior padding. Defaults to **-border** if not specified.

**-sticky** *spec*
Specifies how the image is placed within the final parcel. *spec* contains zero or more characters “n”, “s”, “w”, or “e”.

**-width** *width*
Specifies a minimum width for the element. If less than zero, the base image's width is used as a default.

**IMAGE STRETCHING**

If the element's allocated parcel is larger than the image, the image will be placed in the parcel based on the **-sticky** option. If the image needs to stretch horizontally (i.e., **-sticky ew**) or vertically (**-sticky ns**), subregions of the image are replicated to fill the parcel based on the **-border** option. The **-border** divides the image into 9 regions: four fixed corners, top and left edges (which may be tiled horizontally), left and right edges (which may be tiled vertically), and the central area (which may be tiled in both directions).

**EXAMPLE**
set img1 [image create photo -file button.png]
set img2 [image create photo -file button-pressed.png]
set img3 [image create photo -file button-active.png]

style element create Button.button image \[
    [list $img1 pressed $img2 active $img3] \-
    -border {2 4} -sticky we
]

SEE ALSO

ttk::intro, ttk::style, ttk_vsapi, image, photo

KEYWORDS

style, theme, appearance, pixmap_theme, image

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2004 Joe English
destroy - Destroy one or more windows

SYNOPSIS

destroy ?window window ...?

DESCRIPTION

This command deletes the windows given by the window arguments, plus all of their descendants. If a window "." is deleted then all windows will be destroyed and the application will (normally) exit. The windows are destroyed in order, and if an error occurs in destroying a window the command aborts without destroying the remaining windows. No error is returned if window does not exist.

EXAMPLE

Destroy all checkbuttons that are direct children of the given widget:

```tcl
proc killCheckbuttonChildren {parent} {
    foreach w [winfo children $parent] {
        if {[winfo class $w] eq "Checkbutton"} {
            destroy $w
        }
    }
}
```

KEYWORDS
application, destroy, window

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
spinbox - Create and manipulate spinbox widgets

SYNOPSIS

STANDARD OPTIONS
- activebackground, activeBackground, Foreground
- background or -bg, background, Background
- borderwidth or -bd, borderWidth, BorderWidth
- cursor, cursor, Cursor
- exportselection, exportSelection, ExportSelection
- font, font, Font
- foreground or -fg, foreground, Foreground
- highlightbackground, highlightBackground
- highlightcolor, highlightColor
- highlightthickness, highlightThickness, HighlightThickness
- insertbackground, insertBackground, Foreground
- insertborderwidth, insertBorderWidth, BorderWidth
- insertofftime, insertOffTime, OffTime
- insertontime, insertOnTime, OnTime
- insertwidth, insertWidth, InsertWidth
- justify, justify, Justify
- relief, relief, Relief
- repeatdelay, repeatDelay, RepeatDelay
- repeatinterval, repeatInterval, RepeatInterval
- selectbackground, selectBackground, Foreground
- selectborderwidth, selectBorderWidth, BorderWidth
- selectforeground, selectForeground, Background
- takefocus, takeFocus, TakeFocus
- textvariable, textVariable, Variable
- xscrollcommand, xScrollCommand, ScrollCommand

WIDGET-SPECIFIC OPTIONS
-buttonbackground, buttonBackground, Background
-buttoncursor, buttonCursor, Cursor
-buttondownrelief, buttonDownRelief, Relief
-buttonuprelief, buttonUpRelief, Relief
-command, command, Command
-disabledbackground, disabledBackground,
DisabledBackground
-disabledforeground, disabledForeground, DisabledForeground
-format, format, Format
-from, from, From
-invalidcommand or -vcmd, invalidCommand,
InvalidCommand
-increment, increment, Increment
-readonlybackground, readonlyBackground,
ReadonlyBackground
-state, state, State
-to, to, To
-validate, validate, Validate
-validatecommand or -vcmd, validateCommand,
ValidateCommand
-values, values, Values
-width, width, Width
-wrap, wrap, wrap

DESCRIPTION

VALIDATION

none
focus
focusin
focusout
key
all
%d
%i
%P
%s
%S
%v
WIDGET COMMAND INDICES

number
anchor
del
insert
sel.first
sel.last
@number

SUBCOMMANDS

pathName bbox index
pathName cget option
pathName configure ?option? ?value option value ...?
pathName delete first ?last?
pathName get
pathName icursor index
pathName identify x y
pathName index index
pathName insert index string
pathName invoke element
pathName scan option args
  pathName scan mark x
  pathName scan dragto x
pathName selection option arg
  pathName selection adjust index
  pathName selection clear
  pathName selection element ?element?
  pathName selection from index
  pathName selection present
  pathName selection range start end
  pathName selection to index
pathName set ?string?
pathName validate
pathName xview args
  pathName xview
**NAME**

spinbox - Create and manipulate spinbox widgets

**SYNOPSIS**

`spinbox pathName ?options?`

**STANDARD OPTIONS**

- `-activebackground`, `-activeBackground`, `-Foreground`
- `-background` or `-bg`, `-background`, `-Background`
- `-borderwidth` or `-bd`, `-borderWidth`, `-BorderWidth`
- `-cursor`, `-cursor`, `-Cursor`
- `-exportselection`, `-exportSelection`, `-ExportSelection`
- `-font`, `-font`, `-Font`
- `-foreground` or `-fg`, `-foreground`, `-Foreground`
- `-highlightbackground`, `-highlightBackground`, `-HighlightBackground`
- `-highlightcolor`, `-highlightColor`, `-HighlightColor`
- `-highlightthickness`, `-highlightThickness`, `-HighlightThickness`
- `-insertbackground`, `-insertBackground`, `-Foreground`
- `-insertborderwidth`, `-insertBorderWidth`, `-BorderWidth`
- `-insertofftime`, `-insertOffTime`, `-OffTime`
- `-insertontime`, `-insertOnTime`, `-OnTime`
- `-insertwidth`, `-insertWidth`, `-InsertWidth`
- `-justify`, `-justify`, `-Justify`
- `-relief`, `-relief`, `-Relief`
- `-repeatedelay`, `-repeatDelay`, `-RepeatDelay`
- `-repeatinterval`, `-repeatInterval`, `-RepeatInterval`
- `-selectbackground`, `-selectBackground`, `-Foreground`
- `-selectborderwidth`, `-selectBorderWidth`, `-BorderWidth`
- `-selectforeground`, `-selectForeground`, `-Background`
-takefocus, takeFocus, TakeFocus
-textvariable, textVariable, Variable
-xscrollcommand, xScrollCommand, ScrollCommand

<table>
<thead>
<tr>
<th>WIDGET-SPECIFIC OPTIONS</th>
</tr>
</thead>
</table>

Command-Line Name: **buttonbackground**
Database Name: **buttonBackground**
Database Class: **Background**

The background color to be used for the spin buttons.

Command-Line Name: **buttoncursor**
Database Name: **buttonCursor**
Database Class: **Cursor**

The cursor to be used when over the spin buttons. If this is empty (the default), a default cursor will be used.

Command-Line Name: **buttondownrelief**
Database Name: **buttonDownRelief**
Database Class: **Relief**

The relief to be used for the upper spin button.

Command-Line Name: **buttonuprelief**
Database Name: **buttonUpRelief**
Database Class: **Relief**

The relief to be used for the lower spin button.

Command-Line Name: **command**
Database Name: **command**
Database Class: **Command**

Specifies a Tcl command to invoke whenever a spinbutton is invoked. The command recognizes several percent substitutions: %W for the widget path, %s for the current value of the widget, and %d for the direction of the button pressed (up or down).

Command-Line Name: **disabledbackground**
Database Name: **disabledBackground**
Database Class: **DisabledBackground**
Specifies the background color to use when the spinbox is disabled. If this option is the empty string, the normal background color is used.

Command-Line Name: **-disabledforeground**
Database Name: **disabledForeground**
Database Class: **DisabledForeground**
   Specifies the foreground color to use when the spinbox is disabled. If this option is the empty string, the normal foreground color is used.

Command-Line Name: **-format**
Database Name: **format**
Database Class: **Format**
   Specifies an alternate format to use when setting the string value when using the **-from** and **-to** range. This must be a format specifier of the form `%<pad>.<pad>f`, as it will format a floating-point number.

Command-Line Name: **-from**
Database Name: **from**
Database Class: **From**
   A floating-point value corresponding to the lowest value for a spinbox, to be used in conjunction with **-to** and **-increment**. When all are specified correctly, the spinbox will use these values to control its contents. This value must be less than the **-to** option. If **-values** is specified, it supercedes this option.

Command-Line Name: **-invalidcommand or -invcmd**
Database Name: **invalidCommand**
Database Class: **InvalidCommand**
   Specifies a script to eval when **validateCommand** returns 0. Setting it to an empty string disables this feature (the default). The best use of this option is to set it to **bell**. See **Validation** below for more information.

Command-Line Name: **-increment**
Database Name: **increment**
Database Class: **Increment**
A floating-point value specifying the increment. When used with `-from` and `-to`, the value in the widget will be adjusted by `-increment` when a spin button is pressed (up adds the value, down subtracts the value).

Command-Line Name: **-readonlybackground**
Database Name: **readonlyBackground**
Database Class: **ReadonlyBackground**
Specifies the background color to use when the spinbox is readonly. If this option is the empty string, the normal background color is used.

Command-Line Name: **-state**
Database Name: **state**
Database Class: **State**
Specifies one of three states for the spinbox: *normal*, *disabled*, or *readonly*. If the spinbox is readonly, then the value may not be changed using widget commands and no insertion cursor will be displayed, even if the input focus is in the widget; the contents of the widget may still be selected. If the spinbox is disabled, the value may not be changed, no insertion cursor will be displayed, the contents will not be selectable, and the spinbox may be displayed in a different color, depending on the values of the `-disabledforeground` and `-disabledbackground` options.

Command-Line Name: **-to**
Database Name: **to**
Database Class: **To**
A floating-point value corresponding to the highest value for the spinbox, to be used in conjunction with `-from` and `-increment`. When all are specified correctly, the spinbox will use these values to control its contents. This value must be greater than the `-from` option. If `-values` is specified, it supercedes this option.

Command-Line Name: **-validate**
Database Name: **validate**
Database Class: **Validate**
Specifies the mode in which validation should operate: **none**, **focus**, **focusin**, **focusout**, **key**, or **all**. It defaults to **none**. When you want validation, you must explicitly state which mode you wish to use. See **Validation** below for more.

**Command-Line Name:** `-validatecommand` or `-vcmd`  
**Database Name:** `validateCommand`  
**Database Class:** `ValidateCommand`  
Specifies a script to evaluate when you want to validate the input in the widget. Setting it to an empty string disables this feature (the default). Validation occurs according to the value of **-validate**. This command must return a valid Tcl boolean value. If it returns 0 (or the valid Tcl boolean equivalent) then the value of the widget will not change and the **invalidCommand** will be evaluated if it is set. If it returns 1, then value will be changed. See **Validation** below for more information.

**Command-Line Name:** `-values`  
**Database Name:** `values`  
**Database Class:** `Values`  
Must be a proper list value. If specified, the spinbox will use these values as to control its contents, starting with the first value. This option has precedence over the **-from** and **-to** range.

**Command-Line Name:** `-width`  
**Database Name:** `width`  
**Database Class:** `Width`  
Specifies an integer value indicating the desired width of the spinbox window, in average-size characters of the widget's font. If the value is less than or equal to zero, the widget picks a size just large enough to hold its current text.

**Command-Line Name:** `-wrap`  
**Database Name:** `wrap`  
**Database Class:** `wrap`  
Must be a proper boolean value. If on, the spinbox will wrap around the values of data in the widget.
The **spinbox** command creates a new window (given by the `pathName` argument) and makes it into a spinbox widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the spinbox such as its colors, font, and relief. The **spinbox** command returns its `pathName` argument. At the time this command is invoked, there must not exist a window named `pathName`, but `pathName`'s parent must exist.

A **spinbox** is an extended **entry** widget that allows the user to move, or spin, through a fixed set of ascending or descending values such as times or dates in addition to editing the value as in an **entry**. When first created, a spinbox's string is empty. A portion of the spinbox may be selected as described below. If a spinbox is exporting its selection (see the `exportSelection` option), then it will observe the standard protocols for handling the selection; spinbox selections are available as type **STRING**. Spinboxes also observe the standard Tk rules for dealing with the input focus. When a spinbox has the input focus it displays an **insertion cursor** to indicate where new characters will be inserted.

Spinboxes are capable of displaying strings that are too long to fit entirely within the widget's window. In this case, only a portion of the string will be displayed; commands described below may be used to change the view in the window. Spinboxes use the standard **xScrollCommand** mechanism for interacting with scrollbars (see the description of the **xScrollCommand** option for details). They also support scanning, as described below.

**VALIDATION**

Validation works by setting the **validateCommand** option to a script which will be evaluated according to the **validate** option as follows:

- **none**
  - Default. This means no validation will occur.
- **focus**
**validateCommand** will be called when the spinbox receives or loses focus.

**focusin**
**validateCommand** will be called when the spinbox receives focus.

**focusout**
**validateCommand** will be called when the spinbox loses focus.

**key**
**validateCommand** will be called when the spinbox is edited.

**all**
**validateCommand** will be called for all above conditions.

It is possible to perform percent substitutions on the **validateCommand** and **invalidCommand**, just as you would in a **bind** script. The following substitutions are recognized:

%%%d
Type of action: 1 for **insert**, 0 for **delete**, or -1 for focus, forced or textvariable validation.

%%%i
Index of char string to be inserted/deleted, if any, otherwise -1.

%%%P
The value of the spinbox should edition occur. If you are configuring the spinbox widget to have a new textvariable, this will be the value of that textvariable.

%%%s
The current value of spinbox before edition.

%%%S
The text string being inserted/deleted, if any. Otherwise it is an empty string.

%%%v
The type of validation currently set.

%V
The type of validation that triggered the callback (key, focusin, focusout, forced).

%W
The name of the spinbox widget.

In general, the `textVariable` and `validateCommand` can be dangerous to mix. Any problems have been overcome so that using the `validateCommand` will not interfere with the traditional behavior of the spinbox widget. Using the `textVariable` for read-only purposes will never cause problems. The danger comes when you try set the `textVariable` to something that the `validateCommand` would not accept, which causes `validate` to become `none` (the `invalidCommand` will not be triggered). The same happens when an error occurs evaluating the `validateCommand`.

Primarily, an error will occur when the `validateCommand` or `invalidCommand` encounters an error in its script while evaluating or `validateCommand` does not return a valid Tcl boolean value. The `validate` option will also set itself to `none` when you edit the spinbox widget from within either the `validateCommand` or the `invalidCommand`. Such editions will override the one that was being validated. If you wish to edit the value of the widget during validation and still have the `validate` option set, you should include the command

```
%W config -validate %v
```

in the `validateCommand` or `invalidCommand` (whichever one you were editing the spinbox widget from). It is also recommended to not set an associated `textVariable` during validation, as that can cause the spinbox widget to become out of sync with the `textVariable`.

**WIDGET COMMAND**
The `spinbox` command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

`Option` and the `args` determine the exact behavior of the command.

**INDICES**

Many of the widget commands for spinboxes take one or more indices as arguments. An index specifies a particular character in the spinbox's string, in any of the following ways:

- **number**
  Specifies the character as a numerical index, where 0 corresponds to the first character in the string.

- **anchor**
  Indicates the anchor point for the selection, which is set with the `select from` and `select adjust` widget commands.

- **end**
  Indicates the character just after the last one in the spinbox's string. This is equivalent to specifying a numerical index equal to the length of the spinbox's string.

- **insert**
  Indicates the character adjacent to and immediately following the insertion cursor.

- **sel.first**
  Indicates the first character in the selection. It is an error to use this form if the selection is not in the spinbox window.

- **sel.last**
  Indicates the character just after the last one in the selection. It is
an error to use this form if the selection is not in the spinbox window.

@number
In this form, number is treated as an x-coordinate in the spinbox's window; the character spanning that x-coordinate is used. For example, “@0” indicates the left-most character in the window.

Abbreviations may be used for any of the forms above, e.g. “e” or “sel.f”. In general, out-of-range indices are automatically rounded to the nearest legal value.

SUBCOMMANDS

The following commands are possible for spinbox widgets:

pathName bbox index
Returns a list of four numbers describing the bounding box of the character given by index. The first two elements of the list give the x and y coordinates of the upper-left corner of the screen area covered by the character (in pixels relative to the widget) and the last two elements give the width and height of the character, in pixels. The bounding box may refer to a region outside the visible area of the window.

pathName cget option
Returns the current value of the configuration option given by option. Option may have any of the values accepted by the spinbox command.

pathName configure ?option? ?value option value ...?
Query or modify the configuration options of the widget. If no option is specified, returns a list describing all of the available options for pathName (see Tk_ConfigureInfo for information on the format of this list). If option is specified with no value, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no option is specified). If one or more option-value pairs are specified,
then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the *spinbox* command.

**pathName delete first ?last?**
Delete one or more elements of the spinbox. *First* is the index of the first character to delete, and *last* is the index of the character just after the last one to delete. If *last* is not specified it defaults to *first*+1, i.e. a single character is deleted. This command returns an empty string.

**pathName get**
Returns the spinbox's string.

**pathName icursor index**
Arrange for the insertion cursor to be displayed just before the character given by *index*. Returns an empty string.

**pathName identify x y**
Returns the name of the window element corresponding to coordinates *x* and *y* in the spinbox. Return value is one of: *none*, *buttondown*, *buttonup*, *entry*.

**pathName index index**
Returns the numerical index corresponding to *index*.

**pathName insert index string**
Insert the characters of *string* just before the character indicated by *index*. Returns an empty string.

**pathName invoke element**
Causes the specified element, either *buttondown* or *buttonup*, to be invoked, triggering the action associated with it.

**pathName scan option args**
This command is used to implement scanning on spinboxes. It has two forms, depending on *option*: 
**pathName scan mark x**
Records x and the current view in the spinbox window; used in conjunction with later scan dragto commands. Typically this command is associated with a mouse button press in the widget. It returns an empty string.

**pathName scan dragto x**
This command computes the difference between its x argument and the x argument to the last scan mark command for the widget. It then adjusts the view left or right by 10 times the difference in x-coordinates. This command is typically associated with mouse motion events in the widget, to produce the effect of dragging the spinbox at high speed through the window. The return value is an empty string.

**pathName selection option arg**
This command is used to adjust the selection within a spinbox. It has several forms, depending on option:

**pathName selection adjust index**
Locate the end of the selection nearest to the character given by index, and adjust that end of the selection to be at index (i.e. including but not going beyond index). The other end of the selection is made the anchor point for future select to commands. If the selection is not currently in the spinbox, then a new selection is created to include the characters between index and the most recent selection anchor point, inclusive. Returns an empty string.

**pathName selection clear**
Clear the selection if it is currently in this widget. If the selection is not in this widget then the command has no effect. Returns an empty string.

**pathName selection element ?element?**
Sets or gets the currently selected element. If a spinbutton element is specified, it will be displayed depressed.
**pathName selection from index**
Set the selection anchor point to just before the character given by `index`. Does not change the selection. Returns an empty string.

**pathName selection present**
Returns 1 if there are characters selected in the spinbox, 0 if nothing is selected.

**pathName selection range start end**
Sets the selection to include the characters starting with the one indexed by `start` and ending with the one just before `end`. If `end` refers to the same character as `start` or an earlier one, then the spinbox's selection is cleared.

**pathName selection to index**
If `index` is before the anchor point, set the selection to the characters from `index` up to but not including the anchor point. If `index` is the same as the anchor point, do nothing. If `index` is after the anchor point, set the selection to the characters from the anchor point up to but not including `index`. The anchor point is determined by the most recent `select from` or `select adjust` command in this widget. If the selection is not in this widget then a new selection is created using the most recent anchor point specified for the widget. Returns an empty string.

**pathName set ?string?**
If `string` is specified, the spinbox will try and set it to this value, otherwise it just returns the spinbox's string. If validation is on, it will occur when setting the string.

**pathName validate**
This command is used to force an evaluation of the `validateCommand` independent of the conditions specified by the `validate` option. This is done by temporarily setting the `validate` option to `all`. It returns 0 or 1.

**pathName xview args**
This command is used to query and change the horizontal position of the text in the widget's window. It can take any of the following forms:

**pathName xview**

Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the spinbox's text is off-screen to the left, the middle 40% is visible in the window, and 40% of the text is off-screen to the right. These are the same values passed to scrollbars via the `-xscrollcommand` option.

**pathName xview index**

Adjusts the view in the window so that the character given by `index` is displayed at the left edge of the window.

**pathName xview moveto fraction**

Adjusts the view in the window so that the character `fraction` of the way through the text appears at the left edge of the window. `Fraction` must be a fraction between 0 and 1.

**pathName xview scroll number what**

This command shifts the view in the window left or right according to `number` and `what`. `Number` must be an integer. `What` must be either `units` or `pages` or an abbreviation of one of these. If `what` is `units`, the view adjusts left or right by `number` average-width characters on the display; if it is `pages` then the view adjusts by `number` screenfuls. If `number` is negative then characters farther to the left become visible; if it is positive then characters farther to the right become visible.

---

**DEFAULT BINDINGS**

Tk automatically creates class bindings for spinboxes that give them the following default behavior. In the descriptions below, “word” refers to a contiguous group of letters, digits, or “_” characters, or any single
character other than these.

[1] Clicking mouse button 1 positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget. Dragging with mouse button 1 strokes out a selection between the insertion cursor and the character under the mouse.

[2] Double-clicking with mouse button 1 selects the word under the mouse and positions the insertion cursor at the beginning of the word. Dragging after a double click will stroke out a selection consisting of whole words.

[3] Triple-clicking with mouse button 1 selects all of the text in the spinbox and positions the insertion cursor before the first character.

[4] The ends of the selection can be adjusted by dragging with mouse button 1 while the Shift key is down; this will adjust the end of the selection that was nearest to the mouse cursor when button 1 was pressed. If the button is double-clicked before dragging then the selection will be adjusted in units of whole words.

[5] Clicking mouse button 1 with the Control key down will position the insertion cursor in the spinbox without affecting the selection.

[6] If any normal printing characters are typed in a spinbox, they are inserted at the point of the insertion cursor.

[7] The view in the spinbox can be adjusted by dragging with mouse button 2. If mouse button 2 is clicked without moving the mouse, the selection is copied into the spinbox at the position of the mouse
cursor.

[8] If the mouse is dragged out of the spinbox on the left or right sides while button 1 is pressed, the spinbox will automatically scroll to make more text visible (if there is more text off-screen on the side where the mouse left the window).

[9] The Left and Right keys move the insertion cursor one character to the left or right; they also clear any selection in the spinbox and set the selection anchor. If Left or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character. Control-Left and Control-Right move the insertion cursor by words, and Control-Shift-Left and Control-Shift-Right move the insertion cursor by words and also extend the selection. Control-b and Control-f behave the same as Left and Right, respectively. Meta-b and Meta-f behave the same as Control-Left and Control-Right, respectively.

[10] The Home key, or Control-a, will move the insertion cursor to the beginning of the spinbox and clear any selection in the spinbox. Shift-Home moves the insertion cursor to the beginning of the spinbox and also extends the selection to that point.

[11] The End key, or Control-e, will move the insertion cursor to the end of the spinbox and clear any selection in the spinbox. Shift-End moves the cursor to the end and extends the selection to that point.

[12] The Select key and Control-Space set the selection anchor to the position of the insertion cursor. They do not affect the current selection. Shift-Select and Control-Shift-Space adjust the selection to the current position of the insertion cursor, selecting from the anchor to the insertion cursor if there was not any selection previously.
Control- selects all the text in the spinbox.

Control- clears any selection in the spinbox.

The F16 key (labelled Copy on many Sun workstations) or Meta- copies the selection in the widget to the clipboard, if there is a selection.

The F20 key (labelled Cut on many Sun workstations) or Control- copies the selection in the widget to the clipboard and deletes the selection. If there is no selection in the widget then these keys have no effect.

The F18 key (labelled Paste on many Sun workstations) or Control- inserts the contents of the clipboard at the position of the insertion cursor.

The Delete key deletes the selection, if there is one in the spinbox. If there is no selection, it deletes the character to the right of the insertion cursor.

The BackSpace key and Control-h delete the selection, if there is one in the spinbox. If there is no selection, it deletes the character to the left of the insertion cursor.

Control-d deletes the character to the right of the insertion cursor.

Meta-d deletes the word to the right of the insertion cursor.
Control-k deletes all the characters to the right of the insertion cursor.

[23] Control-t reverses the order of the two characters to the right of the insertion cursor.

If the spinbox is disabled using the -state option, then the spinbox’s view can still be adjusted and text in the spinbox can still be selected, but no insertion cursor will be displayed and no text modifications will take place.

The behavior of spinboxes can be changed by defining new bindings for individual widgets or by redefining the class bindings.

**KEYWORDS**

spinbox, entry, widget
tk_popup - Post a popup menu

SYNOPSIS

tk_popup menu x y ?entry?

DESCRIPTION

This procedure posts a menu at a given position on the screen and configures Tk so that the menu and its cascaded children can be traversed with the mouse or the keyboard. Menu is the name of a menu widget and x and y are the root coordinates at which to display the menu. If entry is omitted or an empty string, the menu's upper left corner is positioned at the given point. Otherwise entry gives the index of an entry in menu and the menu will be positioned so that the entry is positioned over the given point.

EXAMPLE

How to attach a simple popup menu to a widget.

```
# Create a menu
set m [menu .popupMenu]
$m add command -label "Example 1" -command bell
$m add command -label "Example 2" -command bell

# Create something to attach it to
pack [label .l -text "Click me!"]
```
# Arrange for the menu to pop up when the label is clicked
bind .l <1> {tk_popup .popupMenu %X %Y}

SEE ALSO

bind, menu, tk_optionMenu

KEYWORDS

menu, popup

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

ttk::label - Display a text string and/or image

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS

-class
-compound, compound, Compound
-cursor, cursor, Cursor
-image, image, Image
-style
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-width

WIDGET-SPECIFIC OPTIONS

-anchor, anchor, Anchor
-background, frameColor, FrameColor
-font, font, Font
-foreground, textColor, TextColor
-justify, justify, Justify
-padding, padding, Padding
-relief, relief, Relief
-text, text, Text
-wraplength, wrapLength, WrapLength

WIDGET COMMAND

SEE ALSO

NAME

ttk::label - Display a text string and/or image
**SYNOPSIS**

```ttk::label pathName ?options?```

**DESCRIPTION**

A `ttk::label` widget displays a textual label and/or image. The label may be linked to a Tcl variable to automatically change the displayed text.

**STANDARD OPTIONS**

- `-class`
- `-compound, compound, Compound`
- `-cursor, cursor, Cursor`
- `-image, image, Image`
- `-style`
- `-takefocus, takeFocus, TakeFocus`
- `-text, text, Text`
- `-textvariable, textVariable, Variable`
- `-underline, underline, Underline`
- `-width`

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: `-anchor`
Database Name: `anchor`
Database Class: `Anchor`
   Specifies how the information in the widget is positioned relative to the inner margins. Legal values are `n`, `ne`, `e`, `se`, `s`, `sw`, `w`, `nw`, and `center`. See also `-justify`.

Command-Line Name: `-background`
Database Name: `frameColor`
Database Class: `FrameColor`
   The widget's background color. If unspecified, the theme default is used.

Command-Line Name: `-font`
Database Name: **font**
Database Class: **Font**
  Font to use for label text.

Command-Line Name: **-foreground**
Database Name: **textColor**
Database Class: **TextColor**
  The widget's foreground color. If unspecified, the theme default is used.

Command-Line Name: **-justify**
Database Name: **justify**
Database Class: **Justify**
  If there are multiple lines of text, specifies how the lines are laid out relative to one another. One of left, center, or right. See also **-anchor**.

Command-Line Name: **-padding**
Database Name: **padding**
Database Class: **Padding**
  Specifies the amount of extra space to allocate for the widget. The padding is a list of up to four length specifications left top right bottom. If fewer than four elements are specified, bottom defaults to top, right defaults to left, and top defaults to left.

Command-Line Name: **-relief**
Database Name: **relief**
Database Class: **Relief**
  Specifies the 3-D effect desired for the widget border. Valid values are flat, groove, raised, ridge, solid, and sunken.

Command-Line Name: **-text**
Database Name: **text**
Database Class: **Text**
  Specifies a text string to be displayed inside the widget (unless overridden by **-textvariable**).

Command-Line Name: **-wraplength**
Database Name: **wrapLength**
Database Class: **WrapLength**
Specifies the maximum line length (in pixels). If this option is less than or equal to zero, then automatic wrapping is not performed; otherwise the text is split into lines such that no line is longer than the specified value.

**WIDGET COMMAND**

Supports the standard widget commands `configure`, `cget`, `identify`, `instate`, and `state`; see `ttk::widget(n)`.

**SEE ALSO**

`ttk::widget`, `label`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2004 Joe English
NAME
ttk_vsapi - Define a Microsoft Visual Styles element

SYNOPSIS

DESCRIPTION
The vsapi element factory creates a new element in the current theme whose visual appearance is drawn using the Microsoft Visual Styles API which is responsible for the themed styles on Windows XP and Vista. This factory permits any of the Visual Styles parts to be declared as ttk elements that can then be included in a style layout to modify the appearance of ttk widgets.
**className** and **partId** are required parameters and specify the Visual Styles class and part as given in the Microsoft documentation. The **stateMap** may be provided to map ttk states to Visual Styles API states (see **STATE MAP**).

### OPTIONS

Valid options are:

- **-padding padding**
  Specify the element's interior padding. **padding** is a list of up to four integers specifying the left, top, right and bottom padding quantities respectively. This option may not be mixed with any other options.

- **-margins padding**
  Specifies the elements exterior padding. **padding** is a list of up to four integers specifying the left, top, right and bottom padding quantities respectively. This option may not be mixed with any other options.

- **-width width**
  Specifies the height for the element. If this option is set then the Visual Styles API will not be queried for the recommended size or the part. If this option is set then **-height** should also be set. The **-width** and **-height** options cannot be mixed with the **-padding** or **-margins** options.

- **-height height**
  Specifies the height of the element. See the comments for **-width**.

### STATE MAP

The **stateMap** parameter is a list of ttk states and the corresponding Visual Styles API state value. This permits the element appearance to respond to changes in the widget state such as becoming active or being pressed. The list should be as described for the **ttk::style map** command but note that the last pair in the list should be the default state and is typically an empty list and 1. Unfortunately all the Visual
Styles parts have different state values and these must be looked up either in the Microsoft documentation or more likely in the header files. The original header to use was `tmschema.h` but in more recent versions of the Windows Development Kit this is `vssym32.h`.

If no `stateMap` parameter is given there is an implicit default map of `{{}}`

**EXAMPLE**

Create a correctly themed close button by changing the layout of a `ttk::button(n)`. This uses the WINDOW part `WP_SMALLCLOSEBUTTON` and as documented the states `CBS_DISABLED, CBS_HOT, CBS_NORMAL and CBS_PUSHED` are mapped from ttk states.

```plaintext
    ttk::style element create smallclose vsapi WINDOW 19
    {disabled 4 pressed 3 active 2 {} 1}
    ttk::style layout CloseButton {CloseButton.smallclose
    pack [ttk::button .close -style CloseButton]
```

Change the appearance of a `ttk::checkbutton(n)` to use the Explorer pin part `EBP_HEADERPIN`.

```plaintext
    ttk::style element create pin vsapi EXPLORERBAR 3 {
    {pressed !selected} 3
    {active !selected} 2
    {pressed selected} 6
    {active selected} 5
    {selected} 4
    {}
}
    ttk::style layout Explorer.Pin {Explorer.Pin.pin -style
    pack [ttk::checkbutton .pin -style Explorer.Pin]
```
SEE ALSO

ttk::intro, ttk::widget, ttk::style, ttk_image

KEYWORDS

style, theme, appearance, windows

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2008 Pat Thoyts
NAME
entry - Create and manipulate entry widgets
SYNOPSIS
STANDARD OPTIONS
-foreground or -fg, foreground, Foreground
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-exportselection, exportSelection, ExportSelection
-font, font, Font
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-exportselection, exportSelection, ExportSelection
-font, font, Font
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-insertbackground, insertBackground, Foreground
-insertborderwidth, insertBorderWidth, BorderWidth
-insertofftime, insertOffTime, OffTime
-insertonstate, insertOnTime, OnTime
-insertwidth, insertWidth, InsertWidth
-justify, justify, Justify
-relief, relief, Relief
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-takefocus, takeFocus, TakeFocus
-textvariable, textVariable, Variable
-xscrollcommand, xScrollCommand, ScrollCommand
WIDGET-SPECIFIC OPTIONS
-disabledbackground, disabledBackground, DisabledBackground
-disabledforeground, disabledForeground, DisabledForeground
-invalidcommand or -invcmd, invalidCommand, InvalidCommand
-readonlybackground, readonlyBackground, ReadonlyBackground
-show, show, Show
-state, state, State
-validate, validate, Validate
-validatecommand or -vcmd, validateCommand, ValidateCommand
-width, width, Width

DESCRIPTION
VALIDATION
none
focus
focusin
focusout
key
all
%d
%i
%P
%s
%S
%v
%V
%W

WIDGET COMMAND
INDICES
number
anchor
end
insert
sel.first
sel.last
@number

SUBCOMMANDS
pathName bbox index
**NAME**
entry - Create and manipulate entry widgets

**SYNOPSIS**

**entry pathName ?options?**

**STANDARD OPTIONS**

- `-background` or `-bg`, `background`, `Background`
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-exportselection, exportSelection, ExportSelection
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-insertbackground, insertBackground, Foreground
-insertborderwidth, insertBorderWidth, BorderWidth
-insertofftime, insertOffTime, OffTime
-insertontime, insertOnTime, OnTime
-insertwidth, insertWidth, InsertWidth
-justify, justify, Justify
-relief, relief, Relief
-selectbackground, selectBackground, Foreground
-selectborderwidth, selectBorderWidth, BorderWidth
-selectforeground, selectForeground, Background
-takefocus, takeFocus, TakeFocus
-textvariable, textVariable, Variable
-xscrollcommand, xScrollCommand, ScrollCommand

<table>
<thead>
<tr>
<th>WIDGET-SPECIFIC OPTIONS</th>
</tr>
</thead>
</table>

Command-Line Name: **-disabledbackground**
Database Name: **disabledBackground**
Database Class: **DisabledBackground**

Specifies the background color to use when the entry is disabled. If this option is the empty string, the normal background color is used.

Command-Line Name: **-disabledforeground**
Database Name: **disabledForeground**
Database Class: **DisabledForeground**

Specifies the foreground color to use when the entry is disabled. If this option is the empty string, the normal foreground color is used.

Command-Line Name: **-invalidcommand or -invcmd**
Database Name: invalidCommand
Database Class: InvalidCommand
   Specifies a script to eval when validateCommand returns 0. Setting it to {} disables this feature (the default). The best use of this option is to set it to bell. See Validation below for more information.

Command-Line Name: -readonlybackground
Database Name: readonlyBackground
Database Class: ReadonlyBackground
   Specifies the background color to use when the entry is readonly. If this option is the empty string, the normal background color is used.

Command-Line Name: -show
Database Name: show
Database Class: Show
   If this option is specified, then the true contents of the entry are not displayed in the window. Instead, each character in the entry's value will be displayed as the first character in the value of this option, such as “*”. This is useful, for example, if the entry is to be used to enter a password. If characters in the entry are selected and copied elsewhere, the information copied will be what is displayed, not the true contents of the entry.

Command-Line Name: -state
Database Name: state
Database Class: State
   Specifies one of three states for the entry: normal, disabled, or readonly. If the entry is readonly, then the value may not be changed using widget commands and no insertion cursor will be displayed, even if the input focus is in the widget; the contents of the widget may still be selected. If the entry is disabled, the value may not be changed, no insertion cursor will be displayed, the contents will not be selectable, and the entry may be displayed in a different color, depending on the values of the -disabledforeground and -disabledbackground options.
Command-Line Name: **-validate**
Database Name: **validate**
Database Class: **Validate**

Specifies the mode in which validation should operate: none, focus, focusin, focusout, key, or all. It defaults to none. When you want validation, you must explicitly state which mode you wish to use. See **Validation** below for more.

Command-Line Name: **-validatecommand** or **-vcmd**
Database Name: **validateCommand**
Database Class: **ValidateCommand**

Specifies a script to eval when you want to validate the input into the entry widget. Setting it to {} disables this feature (the default). This command must return a valid Tcl boolean value. If it returns 0 (or the valid Tcl boolean equivalent) then it means you reject the new edition and it will not occur and the invalidCommand will be evaluated if it is set. If it returns 1, then the new edition occurs. See **Validation** below for more information.

Command-Line Name: **-width**
Database Name: **width**
Database Class: **Width**

Specifies an integer value indicating the desired width of the entry window, in average-size characters of the widget's font. If the value is less than or equal to zero, the widget picks a size just large enough to hold its current text.

**DESCRIPTION**

The **entry** command creates a new window (given by the **pathName** argument) and makes it into an entry widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the entry such as its colors, font, and relief. The **entry** command returns its **pathName** argument. At the time this command is invoked, there must not exist a window named **pathName**, but **pathName**'s parent must exist.

An entry is a widget that displays a one-line text string and allows that
string to be edited using widget commands described below, which are typically bound to keystrokes and mouse actions. When first created, an entry's string is empty. A portion of the entry may be selected as described below. If an entry is exporting its selection (see the exportSelection option), then it will observe the standard X11 protocols for handling the selection; entry selections are available as type STRING. Entries also observe the standard Tk rules for dealing with the input focus. When an entry has the input focus it displays an insertion cursor to indicate where new characters will be inserted.

Entries are capable of displaying strings that are too long to fit entirely within the widget's window. In this case, only a portion of the string will be displayed; commands described below may be used to change the view in the window. Entries use the standard xScrollCommand mechanism for interacting with scrollbars (see the description of the xScrollCommand option for details). They also support scanning, as described below.

**VALIDATION**

Validation works by setting the validateCommand option to a script which will be evaluated according to the validate option as follows:

- **none**
  - Default. This means no validation will occur.

- **focus**
  - validateCommand will be called when the entry receives or loses focus.

- **focusin**
  - validateCommand will be called when the entry receives focus.

- **focusout**
  - validateCommand will be called when the entry loses focus.

- **key**
  - validateCommand will be called when the entry is edited.
all

validateCommand will be called for all above conditions.

It is possible to perform percent substitutions on the validateCommand and invalidCommand, just as you would in a bind script. The following substitutions are recognized:

%d
Type of action: 1 for insert, 0 for delete, or -1 for focus, forced or textvariable validation.

%i
Index of char string to be inserted/deleted, if any, otherwise -1.

%P
The value of the entry if the edit is allowed. If you are configuring the entry widget to have a new textvariable, this will be the value of that textvariable.

%s
The current value of entry prior to editing.

%S
The text string being inserted/deleted, if any, {} otherwise.

%v
The type of validation currently set.

%V
The type of validation that triggered the callback (key, focusin, focusout, forced).

%W
The name of the entry widget.

In general, the textVariable and validateCommand can be dangerous to mix. Any problems have been overcome so that using the validateCommand will not interfere with the traditional behavior of the entry widget. Using the textVariable for read-only purposes will never
cause problems. The danger comes when you try set the `textVariable` to something that the `validateCommand` would not accept, which causes `validate` to become `none` (the `invalidCommand` will not be triggered). The same happens when an error occurs evaluating the `validateCommand`.

Primarily, an error will occur when the `validateCommand` or `invalidCommand` encounters an error in its script while evaluating or `validateCommand` does not return a valid Tcl boolean value. The `validate` option will also set itself to `none` when you edit the entry widget from within either the `validateCommand` or the `invalidCommand`. Such editions will override the one that was being validated. If you wish to edit the entry widget (for example set it to `{}`) during validation and still have the `validate` option set, you should include the command

```
after idle {%W config -validate %v}
```

in the `validateCommand` or `invalidCommand` (whichever one you were editing the entry widget from). It is also recommended to not set an associated `textVariable` during validation, as that can cause the entry widget to become out of sync with the `textVariable`.

**WIDGET COMMAND**

The `entry` command creates a new Tcl command whose name is `pathName`. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName subcommand ?arg arg ...?
```

Subcommand and the args determine the exact behavior of the command.
Many of the widget commands for entries take one or more indices as arguments. An index specifies a particular character in the entry's string, in any of the following ways:

*number*

Specifies the character as a numerical index, where 0 corresponds to the first character in the string.

*anchor*

Indicates the anchor point for the selection, which is set with the `select from` and `select adjust` widget commands.

*end*

Indicates the character just after the last one in the entry's string. This is equivalent to specifying a numerical index equal to the length of the entry's string.

*insert*

Indicates the character adjacent to and immediately following the insertion cursor.

*sel.first*

Indicates the first character in the selection. It is an error to use this form if the selection is not in the entry window.

*sel.last*

Indicates the character just after the last one in the selection. It is an error to use this form if the selection is not in the entry window.

*@number*

In this form, `number` is treated as an x-coordinate in the entry's window; the character spanning that x-coordinate is used. For example, “@0” indicates the left-most character in the window.

Abbreviations may be used for any of the forms above, e.g. “e” or “sel.f”. In general, out-of-range indices are automatically rounded to the nearest legal value.
The following commands are possible for entry widgets:

`pathName bbox index`
Returns a list of four numbers describing the bounding box of the character given by `index`. The first two elements of the list give the x and y coordinates of the upper-left corner of the screen area covered by the character (in pixels relative to the widget) and the last two elements give the width and height of the character, in pixels. The bounding box may refer to a region outside the visible area of the window.

`pathName cget option`
Returns the current value of the configuration option given by `option`. `Option` may have any of the values accepted by the `entry` command.

`pathName configure ?option? ?value option value ...?`
Query or modify the configuration options of the widget. If no `option` is specified, returns a list describing all of the available options for `pathName` (see `Tk_ConfigureInfo` for information on the format of this list). If `option` is specified with no `value`, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no `option` is specified). If one or more `option-value` pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. `Option` may have any of the values accepted by the `entry` command.

`pathName delete first ?last?`
Delete one or more elements of the entry. `First` is the index of the first character to delete, and `last` is the index of the character just after the last one to delete. If `last` is not specified it defaults to `first+1`, i.e. a single character is deleted. This command returns an empty string.
**pathName get**
Returns the entry's string.

**pathName icursor index**
Arrange for the insertion cursor to be displayed just before the character given by `index`. Returns an empty string.

**pathName index index**
Returns the numerical index corresponding to `index`.

**pathName insert index string**
Insert the characters of `string` just before the character indicated by `index`. Returns an empty string.

**pathName scan option args**
This command is used to implement scanning on entries. It has two forms, depending on `option`:

**pathName scan mark x**
Records `x` and the current view in the entry window; used in conjunction with later `scan dragto` commands. Typically this command is associated with a mouse button press in the widget. It returns an empty string.

**pathName scan dragto x**
This command computes the difference between its `x` argument and the `x` argument to the last `scan mark` command for the widget. It then adjusts the view left or right by 10 times the difference in x-coordinates. This command is typically associated with mouse motion events in the widget, to produce the effect of dragging the entry at high speed through the window. The return value is an empty string.

**pathName selection option arg**
This command is used to adjust the selection within an entry. It has several forms, depending on `option`:

**pathName selection adjust index**
Locate the end of the selection nearest to the character given by *index*, and adjust that end of the selection to be at *index* (i.e. including but not going beyond *index*). The other end of the selection is made the anchor point for future `select to` commands. If the selection is not currently in the entry, then a new selection is created to include the characters between *index* and the most recent selection anchor point, inclusive. Returns an empty string.

```
pathName selection clear
```
Clear the selection if it is currently in this widget. If the selection is not in this widget then the command has no effect. Returns an empty string.

```
pathName selection from index
```
Set the selection anchor point to just before the character given by *index*. Does not change the selection. Returns an empty string.

```
pathName selection present
```
Returns 1 if there is a characters selected in the entry, 0 if nothing is selected.

```
pathName selection range start end
```
Sets the selection to include the characters starting with the one indexed by *start* and ending with the one just before *end*. If *end* refers to the same character as *start* or an earlier one, then the entry's selection is cleared.

```
pathName selection to index
```
If *index* is before the anchor point, set the selection to the characters from *index* up to but not including the anchor point. If *index* is the same as the anchor point, do nothing. If *index* is after the anchor point, set the selection to the characters from the anchor point up to but not including *index*. The anchor point is determined by the most recent `select from` or `select adjust` command in this widget. If the selection is not in this widget then a new selection is created using the most recent
anchor point specified for the widget. Returns an empty string.

**pathName validate**

This command is used to force an evaluation of the validateCommand independent of the conditions specified by the validate option. This is done by temporarily setting the validate option to all. It returns 0 or 1.

**pathName xview args**

This command is used to query and change the horizontal position of the text in the widget's window. It can take any of the following forms:

**pathName xview**

Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the entry's text is off-screen to the left, the middle 40% is visible in the window, and 40% of the text is off-screen to the right. These are the same values passed to scrollbars via the -xscrollcommand option.

**pathName xview index**

Adjusts the view in the window so that the character given by index is displayed at the left edge of the window.

**pathName xview moveto fraction**

Adjusts the view in the window so that the character fraction of the way through the text appears at the left edge of the window. Fraction must be a fraction between 0 and 1.

**pathName xview scroll number what**

This command shifts the view in the window left or right according to number and what. Number must be an integer. What must be either units or pages or an abbreviation of one of these. If what is units, the view adjusts left or right by number average-width characters on the display; if it is pages
then the view adjusts by number screenfuls. If number is negative then characters farther to the left become visible; if it is positive then characters farther to the right become visible.

**DEFAULT BINDINGS**

Tk automatically creates class bindings for entries that give them the following default behavior. In the descriptions below, “word” refers to a contiguous group of letters, digits, or “_” characters, or any single character other than these.

[1] Clicking mouse button 1 positions the insertion cursor just before the character underneath the mouse cursor, sets the input focus to this widget, and clears any selection in the widget. Dragging with mouse button 1 strokes out a selection between the insertion cursor and the character under the mouse.

[2] Double-clicking with mouse button 1 selects the word under the mouse and positions the insertion cursor at the end of the word. Dragging after a double click will stroke out a selection consisting of whole words.

[3] Triple-clicking with mouse button 1 selects all of the text in the entry and positions the insertion cursor at the end of the line.

[4] The ends of the selection can be adjusted by dragging with mouse button 1 while the Shift key is down; this will adjust the end of the selection that was nearest to the mouse cursor when button 1 was pressed. If the button is double-clicked before dragging then the selection will be adjusted in units of whole words.

[5] Clicking mouse button 1 with the Control key down will position the insertion cursor in the entry without affecting the selection.
If any normal printing characters are typed in an entry, they are inserted at the point of the insertion cursor.

The view in the entry can be adjusted by dragging with mouse button 2. If mouse button 2 is clicked without moving the mouse, the selection is copied into the entry at the position of the mouse cursor.

If the mouse is dragged out of the entry on the left or right sides while button 1 is pressed, the entry will automatically scroll to make more text visible (if there is more text off-screen on the side where the mouse left the window).

The Left and Right keys move the insertion cursor one character to the left or right; they also clear any selection in the entry and set the selection anchor. If Left or Right is typed with the Shift key down, then the insertion cursor moves and the selection is extended to include the new character. Control-Left and Control-Right move the insertion cursor by words, and Control-Shift-Left and Control-Shift-Right move the insertion cursor by words and also extend the selection. Control-b and Control-f behave the same as Left and Right, respectively. Meta-b and Meta-f behave the same as Control-Left and Control-Right, respectively.

The Home key, or Control-a, will move the insertion cursor to the beginning of the entry and clear any selection in the entry. Shift-Home moves the insertion cursor to the beginning of the entry and also extends the selection to that point.

The End key, or Control-e, will move the insertion cursor to the end of the entry and clear any selection in the entry. Shift-End moves the cursor to the end and extends the selection to that point.
The Select key and Control-Space set the selection anchor to the position of the insertion cursor. They do not affect the current selection. Shift-Select and Control-Shift-Space adjust the selection to the current position of the insertion cursor, selecting from the anchor to the insertion cursor if there was not any selection previously.

Control-\ selects all the text in the entry.

Control-\ clears any selection in the entry.

The F16 key (labelled Copy on many Sun workstations) or Meta-w copies the selection in the widget to the clipboard, if there is a selection.

The F20 key (labelled Cut on many Sun workstations) or Control-w copies the selection in the widget to the clipboard and deletes the selection. If there is no selection in the widget then these keys have no effect.

The F18 key (labelled Paste on many Sun workstations) or Control-y inserts the contents of the clipboard at the position of the insertion cursor.

The Delete key deletes the selection, if there is one in the entry. If there is no selection, it deletes the character to the right of the insertion cursor.

The BackSpace key and Control-h delete the selection, if there is one in the entry. If there is no selection, it deletes the character to
the left of the insertion cursor.

[20]
Control-d deletes the character to the right of the insertion cursor.

[21]
Meta-d deletes the word to the right of the insertion cursor.

[22]
Control-k deletes all the characters to the right of the insertion cursor.

[23]
Control-t reverses the order of the two characters to the right of the insertion cursor.

If the entry is disabled using the -state option, then the entry's view can still be adjusted and text in the entry can still be selected, but no insertion cursor will be displayed and no text modifications will take place except if the entry is linked to a variable using the -textvariable option, in which case any changes to the variable are reflected by the entry whatever the value of its -state option.

The behavior of entries can be changed by defining new bindings for individual widgets or by redefining the class bindings.

SEE ALSO

ttk::entry

KEYWORDS

entry, widget

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
menubutton - Create and manipulate menubutton widgets

SYNOPSIS

STANDARD OPTIONS
-activebackground, activeBackground, Foreground
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

WIDGET-SPECIFIC OPTIONS
-direction, direction, Height
-height, height, Height
-indicatoron, indicatorOn, IndicatorOn
-menu, menu, MenuName
-state, state, State
-width, width, Width

INTRODUCTION

WIDGET COMMAND

`pathName cget option`

`pathName configure ?option? ?value option value ...?`

DEFAULT BINDINGS

SEE ALSO

KEYWORDS

NAME

menubutton - Create and manipulate menubutton widgets

SYNOPSIS

`menubutton pathName ?options?`

STANDARD OPTIONS

-activebackground, activeBackground, Foreground
-activeforeground, activeForeground, Background
-anchor, anchor, Anchor
-background or -bg, background, Background
-bitmap, bitmap, Bitmap
-borderwidth or -bd, borderWidth, BorderWidth
-compound, compound, Compound
-cursor, cursor, Cursor
-disabledforeground, disabledForeground, DisabledForeground
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-image, image, Image
-justify, justify, Justify
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable
-underline, underline, Underline
-wraplength, wrapLength, WrapLength

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: **-direction**
Database Name: **direction**
Database Class: **Height**
   Specifies where the menu is going to be popup up. **above** tries to pop the menu above the menubutton. **below** tries to pop the menu below the menubutton. **left** tries to pop the menu to the left of the menubutton. **right** tries to pop the menu to the right of the menubutton. **flush** pops the menu directly over the menubutton. In the case of **above** or **below**, the direction will be reversed if the menu would show offscreen.

Command-Line Name: **-height**
Database Name: **height**
Database Class: **Height**
   Specifies a desired height for the menubutton. If an image or bitmap is being displayed in the menubutton then the value is in screen units (i.e. any of the forms acceptable to **Tk_GetPixels**); for text it is in lines of text. If this option is not specified, the menubutton’s desired height is computed from the size of the image or bitmap or text being displayed in it.

Command-Line Name: **-indicatoron**
Database Name: **indicatorOn**
Database Class: **IndicatorOn**
   The value must be a proper boolean value. If it is true then a small indicator rectangle will be displayed on the right side of the
menubutton and the default menu bindings will treat this as an option menubutton. If false then no indicator will be displayed.

Command-Line Name: `-menu`
Database Name: `menu`
Database Class: `MenuName`
  Specifies the path name of the menu associated with this menubutton. The menu must be a child of the menubutton.

Command-Line Name: `-state`
Database Name: `state`
Database Class: `State`
  Specifies one of three states for the menubutton: normal, active, or disabled. In normal state the menubutton is displayed using the foreground and background options. The active state is typically used when the pointer is over the menubutton. In active state the menubutton is displayed using the activeForeground and activeBackground options. Disabled state means that the menubutton should be insensitive: the default bindings will refuse to activate the widget and will ignore mouse button presses. In this state the disabledForeground and background options determine how the button is displayed.

Command-Line Name: `-width`
Database Name: `width`
Database Class: `Width`
  Specifies a desired width for the menubutton. If an image or bitmap is being displayed in the menubutton then the value is in screen units (i.e. any of the forms acceptable to `Tk_GetPixels`); for text it is in characters. If this option is not specified, the menubutton's desired width is computed from the size of the image or bitmap or text being displayed in it.

INTRODUCTION

The menubutton command creates a new window (given by the pathName argument) and makes it into a menubutton widget. Additional options, described above, may be specified on the command line or in
the option database to configure aspects of the menubutton such as its colors, font, text, and initial relief. The menubutton command returns its pathName argument. At the time this command is invoked, there must not exist a window named pathName, but pathName's parent must exist.

A menubutton is a widget that displays a textual string, bitmap, or image and is associated with a menu widget. If text is displayed, it must all be in a single font, but it can occupy multiple lines on the screen (if it contains newlines or if wrapping occurs because of the wrapLength option) and one of the characters may optionally be underlined using the underline option. In normal usage, pressing mouse button 1 over the menubutton causes the associated menu to be posted just underneath the menubutton. If the mouse is moved over the menu before releasing the mouse button, the button release causes the underlying menu entry to be invoked. When the button is released, the menu is unposted.

Menubuttons are typically organized into groups called menu bars that allow scanning: if the mouse button is pressed over one menubutton (causing it to post its menu) and the mouse is moved over another menubutton in the same menu bar without releasing the mouse button, then the menu of the first menubutton is unposted and the menu of the new menubutton is posted instead.

There are several interactions between menubuttons and menus; see the menu manual entry for information on various menu configurations, such as pulldown menus and option menus.

**WIDGET COMMAND**

The menubutton command creates a new Tcl command whose name is pathName. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```
Option and the args determine the exact behavior of the command. The following commands are possible for menubutton widgets:

\textit{pathName cget option}

Returns the current value of the configuration option given by \textit{option}. \textit{Option} may have any of the values accepted by the \textit{menubutton} command.

\textit{pathName configure ?option? ?value option value ...?}

Query or modify the configuration options of the widget. If no \textit{option} is specified, returns a list describing all of the available options for \textit{pathName} (see Tk\_ConfigureInfo for information on the format of this list). If \textit{option} is specified with no \textit{value}, then the command returns a list describing the one named \textit{option} (this list will be identical to the corresponding sublist of the value returned if no \textit{option} is specified). If one or more \textit{option-value} pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. \textit{Option} may have any of the values accepted by the \textit{menubutton} command.

\textbf{DEFAULT BINDINGS}

Tk automatically creates class bindings for menubuttons that give them the following default behavior:

[1] A menubutton activates whenever the mouse passes over it and deactivates whenever the mouse leaves it.

[2] Pressing mouse button 1 over a menubutton posts the menubutton: its relief changes to raised and its associated menu is posted under the menubutton. If the mouse is dragged down into the menu with the button still down, and if the mouse button is then released over an entry in the menu, the menubutton is unposted and the menu entry is invoked.
If button 1 is pressed over a menubutton and then released over that menubutton, the menubutton stays posted: you can still move the mouse over the menu and click button 1 on an entry to invoke it. Once a menu entry has been invoked, the menubutton unposts itself.

If button 1 is pressed over a menubutton and then dragged over some other menubutton, the original menubutton unposts itself and the new menubutton posts.

If button 1 is pressed over a menubutton and released outside any menubutton or menu, the menubutton unposts without invoking any menu entry.

When a menubutton is posted, its associated menu claims the input focus to allow keyboard traversal of the menu and its submenus. See the menu manual entry for details on these bindings.

If the underline option has been specified for a menubutton then keyboard traversal may be used to post the menubutton: Alt+\textit{x}, where \textit{x} is the underlined character (or its lower-case or upper-case equivalent), may be typed in any window under the menubutton’s toplevel to post the menubutton.

The F10 key may be typed in any window to post the first menubutton under its toplevel window that is not disabled.

If a menubutton has the input focus, the space and return keys post the menubutton.

If the menubutton’s state is \textbf{disabled} then none of the above actions
occur: the menubutton is completely non-responsive.

The behavior of menubuttons can be changed by defining new bindings for individual widgets or by redefining the class bindings.

SEE ALSO

`ttk::menubutton`, `menu`

KEYWORDS

`menubutton`, `widget`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
tk_setPalette, tk_bisque - Modify the Tk color palette

SYNOPSIS

```
tk_setPalette background
```
```
tk_setPalette name value ?name value ...? 
```
```
tk_bisque
```

DESCRIPTION

The `tk_setPalette` procedure changes the color scheme for Tk. It does this by modifying the colors of existing widgets and by changing the option database so that future widgets will use the new color scheme. If `tk_setPalette` is invoked with a single argument, the argument is the name of a color to use as the normal background color; `tk_setPalette` will compute a complete color palette from this background color. Alternatively, the arguments to `tk_setPalette` may consist of any number of `name-value` pairs, where the first argument of the pair is the name of an option in the Tk option database and the second argument is the new value to use for that option. The following database names are currently supported:

- `activeBackground`
- `foreground`
- `selectColor`
- `activeForeground`
- `highlightBackgroundColor`
- `selectBackgroundColor`
- `background`
- `highlightColor`
- `selectForegroundColor`
tk_setPalette tries to compute reasonable defaults for any options that you do not specify. You can specify options other than the above ones and Tk will change those options on widgets as well. This feature may be useful if you are using custom widgets with additional color options.

Once it has computed the new value to use for each of the color options, tk_setPalette scans the widget hierarchy to modify the options of all existing widgets. For each widget, it checks to see if any of the above options is defined for the widget. If so, and if the option's current value is the default, then the value is changed; if the option has a value other than the default, tk_setPalette will not change it. The default for an option is the one provided by the widget (\[index \$w configure \$option 3\]) unless tk_setPalette has been run previously, in which case it is the value specified in the previous invocation of tk_setPalette.

After modifying all the widgets in the application, tk_setPalette adds options to the option database to change the defaults for widgets created in the future. The new options are added at priority widgetDefault, so they will be overridden by options from the .Xdefaults file or options specified on the command-line that creates a widget.

The procedure tk_bisque is provided for backward compatibility: it restores the application's colors to the light brown ("bisque") color scheme used in Tk 3.6 and earlier versions.

KEYWORDS

bisque, color, palette
NAME

ttk::labelframe - Container widget with optional label

SYNOPSIS

description

STANDARD OPTIONS

-class
-cursor, cursor, Cursor
-style
-takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS

-labelanchor, labelAnchor, LabelAnchor
-text, text, Text
-underline, underline, Underline
-padding, padding, Padding
-labelwidget, labelWidget, LabelWidget
-width, width, Width
-height, height, Height

WIDGET COMMAND

SEE ALSO

KEYWORDS

NAME

ttk::labelframe - Container widget with optional label

SYNOPSIS

description

A ttk::labelframe widget is a container used to group other widgets
together. It has an optional label, which may be a plain text string or another widget.

**STANDARD OPTIONS**

- `-class`
- `-cursor`, `cursor`, `Cursor`
- `-style`
- `-takefocus`, `takeFocus`, `TakeFocus`

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: `-labelanchor`
Database Name: `labelAnchor`
Database Class: `LabelAnchor`
    Specifies where to place the label. Allowed values are (clockwise from the top upper left corner): `nw`, `n`, `ne`, `en`, `e`, `es`, `se`, `s`, `sw`, `ws`, `w` and `wn`. The default value is theme-dependent.

Command-Line Name: `-text`
Database Name: `text`
Database Class: `Text`
    Specifies the text of the label.

Command-Line Name: `-underline`
Database Name: `underline`
Database Class: `Underline`
    If set, specifies the integer index (0-based) of a character to underline in the text string. The underlined character is used for mnemonic activation. Mnemonic activation for a `ttk::labelframe` sets the keyboard focus to the first child of the `ttk::labelframe` widget.

Command-Line Name: `-padding`
Database Name: `padding`
Database Class: `Padding`
    Additional padding to include inside the border.
Command-Line Name: -labelwidget
Database Name: labelWidget
Database Class: LabelWidget
   The name of a widget to use for the label. If set, overrides the -text option. The -labelwidget must be a child of the labelframe widget or one of the labelframe's ancestors, and must belong to the same top-level widget as the labelframe.

Command-Line Name: -width
Database Name: width
Database Class: Width
   If specified, the widget's requested width in pixels.

Command-Line Name: -height
Database Name: height
Database Class: Height
   If specified, the widget's requested height in pixels. (See ttk::frame(n) for further notes on -width and -height).

## WIDGET COMMAND

Supports the standard widget commands configure, cget, identify, instate, and state; see ttk::widget(n).

## SEE ALSO

ttk::widget, ttk::frame, labelframe

## KEYWORDS

widget, frame, container, label, groupbox

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2005 Joe English
NAME
winfo - Return window-related information

SYNOPSIS

DESCRIPTION

winfo atom ?-displayof window? name
winfo atomname ?-displayof window? id
winfo cells window
winfo children window
winfo class window
winfo colormapfull window
winfo containing ?-displayof window? rootX rootY
winfo depth window
winfo exists window
winfo fpixels window number
winfo geometry window
winfo height window
winfo id window
winfo interps ?-displayof window?
winfo ismapped window
winfo manager window
winfo name window
winfo parent window
winfo pathname ?-displayof window? id
winfo pixels window number
winfo pointerx window
winfo pointerxy window
winfo pointery window
winfo reqheight window
winfo reqwidth window
winfo rgb window color
winfo rootx window
The `winfo` command is used to retrieve information about windows managed by Tk. It can take any of a number of different forms, depending on the `option` argument. The legal forms are:
**winfo atom** ?-*displayof* *window*? *name*

Returns a decimal string giving the integer identifier for the atom whose name is *name*. If no atom exists with the name *name* then a new one is created. If the -*displayof* option is given then the atom is looked up on the display of *window*; otherwise it is looked up on the display of the application's main window.

**winfo atomname** ?-*displayof* *window*? *id*

Returns the textual name for the atom whose integer identifier is *id*. If the -*displayof* option is given then the identifier is looked up on the display of *window*; otherwise it is looked up on the display of the application's main window. This command is the inverse of the **winfo atom** command. It generates an error if no such atom exists.

**winfo cells** *window*

Returns a decimal string giving the number of cells in the color map for *window*.

**winfo children** *window*

Returns a list containing the path names of all the children of *window*. Top-level windows are returned as children of their logical parents. The list is in stacking order, with the lowest window first, except for Top-level windows which are not returned in stacking order. Use the **wm stackorder** command to query the stacking order of Top-level windows.

**winfo class** *window*

Returns the class name for *window*.

**winfo colormapfull** *window*

Returns 1 if the colormap for *window* is known to be full, 0 otherwise. The colormap for a window is “known” to be full if the last attempt to allocate a new color on that window failed and this application has not freed any colors in the colormap since the failed allocation.

**winfo containing** ?-*displayof* *window*? *rootX* *rootY*

Returns the path name for the window containing the point given by
rootX and rootY. RootX and rootY are specified in screen units (i.e. any form acceptable to Tk._GetPixels) in the coordinate system of the root window (if a virtual-root window manager is in use then the coordinate system of the virtual root window is used). If the -displayof option is given then the coordinates refer to the screen containing window; otherwise they refer to the screen of the application's main window. If no window in this application contains the point then an empty string is returned. In selecting the containing window, children are given higher priority than parents and among siblings the highest one in the stacking order is chosen.

winfo depth window
Returns a decimal string giving the depth of window (number of bits per pixel).

winfo exists window
Returns 1 if there exists a window named window, 0 if no such window exists.

winfo fpixels window number
Returns a floating-point value giving the number of pixels in window corresponding to the distance given by number. Number may be specified in any of the forms acceptable to Tk._GetScreenMM, such as “2.0c” or “1i”. The return value may be fractional; for an integer value, use winfo pixels.

winfo geometry window
Returns the geometry for window, in the form widthxheight+x+y. All dimensions are in pixels.

winfo height window
Returns a decimal string giving window's height in pixels. When a window is first created its height will be 1 pixel; the height will eventually be changed by a geometry manager to fulfill the window's needs. If you need the true height immediately after creating a widget, invoke update to force the geometry manager to arrange it, or use winfo reqheight to get the window's requested height instead of its actual height.
winfo id *window*
Returns a hexadecimal string giving a low-level platform-specific identifier for *window*. On Unix platforms, this is the X window identifier. Under Windows, this is the Windows HWND. On the Macintosh the value has no meaning outside Tk.

winfo interps ?-displayof *window*?
Returns a list whose members are the names of all Tcl interpreters (e.g. all Tk-based applications) currently registered for a particular display. If the -displayof option is given then the return value refers to the display of *window*; otherwise it refers to the display of the application's main window.

winfo ismapped *window*
Returns 1 if *window* is currently mapped, 0 otherwise.

winfo manager *window*
Returns the name of the geometry manager currently responsible for *window*, or an empty string if *window* is not managed by any geometry manager. The name is usually the name of the Tcl command for the geometry manager, such as pack or place. If the geometry manager is a widget, such as canvases or text, the name is the widget's class command, such as canvas.

winfo name *window*
Returns *window*'s name (i.e. its name within its parent, as opposed to its full path name). The command winfo name . will return the name of the application.

winfo parent *window*
Returns the path name of *window*'s parent, or an empty string if *window* is the main window of the application.

winfo pathname ?-displayof *window*? *id*
Returns the path name of the window whose X identifier is *id*. *Id* must be a decimal, hexadecimal, or octal integer and must correspond to a window in the invoking application. If the -displayof option is given then the identifier is looked up on the
display of window; otherwise it is looked up on the display of the application's main window.

**winfo pixels** *window number*

Returns the number of pixels in window corresponding to the distance given by number. *Number* may be specified in any of the forms acceptable to Tk::GetPixels, such as “2.0c” or “1i”. The result is rounded to the nearest integer value; for a fractional result, use winfo fpixels.

**winfo pointerx** *window*

If the mouse pointer is on the same screen as window, returns the pointer's x coordinate, measured in pixels in the screen's root window. If a virtual root window is in use on the screen, the position is measured in the virtual root. If the mouse pointer is not on the same screen as window then -1 is returned.

**winfo pointerxy** *window*

If the mouse pointer is on the same screen as window, returns a list with two elements, which are the pointer's x and y coordinates measured in pixels in the screen's root window. If a virtual root window is in use on the screen, the position is computed in the virtual root. If the mouse pointer is not on the same screen as window then both of the returned coordinates are -1.

**winfo pointery** *window*

If the mouse pointer is on the same screen as window, returns the pointer's y coordinate, measured in pixels in the screen's root window. If a virtual root window is in use on the screen, the position is computed in the virtual root. If the mouse pointer is not on the same screen as window then -1 is returned.

**winfo reqheight** *window*

Returns a decimal string giving window's requested height, in pixels. This is the value used by window's geometry manager to compute its geometry.

**winfo reqwidth** *window*
Returns a decimal string giving window's requested width, in pixels. This is the value used by window's geometry manager to compute its geometry.

**winfo rgb window color**
Returns a list containing three decimal values in the range 0 to 65535, which are the red, green, and blue intensities that correspond to color in the window given by window. Color may be specified in any of the forms acceptable for a color option.

**winfo rootx window**
Returns a decimal string giving the x-coordinate, in the root window of the screen, of the upper-left corner of window's border (or window if it has no border).

**winfo rooty window**
Returns a decimal string giving the y-coordinate, in the root window of the screen, of the upper-left corner of window's border (or window if it has no border).

**winfo screen window**
Returns the name of the screen associated with window, in the form displayName.screenIndex.

**winfo screencells window**
Returns a decimal string giving the number of cells in the default color map for window's screen.

**winfo screendepth window**
Returns a decimal string giving the depth of the root window of window's screen (number of bits per pixel).

**winfo screenheight window**
Returns a decimal string giving the height of window's screen, in pixels.

**winfo screenmmheight window**
Returns a decimal string giving the height of window's screen, in
millimeters.

**winfo screenmmwidth** *window*

Returns a decimal string giving the width of *window*’s screen, in millimeters.

**winfo screenvisual** *window*

Returns one of the following strings to indicate the default visual class for *window*’s screen: **directcolor**, **grayscale**, **pseudocolor**, **staticcolor**, **staticgray**, or **truecolor**.

**winfo screenwidth** *window*

Returns a decimal string giving the width of *window*’s screen, in pixels.

**winfo server** *window*

Returns a string containing information about the server for *window*’s display. The exact format of this string may vary from platform to platform. For X servers the string has the form “**XmajorRminor vendor vendorVersion**” where *major* and *minor* are the version and revision numbers provided by the server (e.g., **X11R5**), *vendor* is the name of the vendor for the server, and *vendorRelease* is an integer release number provided by the server.

**winfo toplevel** *window*

Returns the path name of the top-of-hierarchy window containing *window*. In standard Tk this will always be a **toplevel** widget, but extensions may create other kinds of top-of-hierarchy widgets.

**winfo viewable** *window*

Returns 1 if *window* and all of its ancestors up through the nearest toplevel window are mapped. Returns 0 if any of these windows are not mapped.

**winfo visual** *window*

Returns one of the following strings to indicate the visual class for *window*: **directcolor**, **grayscale**, **pseudocolor**, **staticcolor**, **staticgray**, or **truecolor**.
staticgray, or truecolor.

winfo visualid window
Returns the X identifier for the visual for window.

winfo visualsavailable window ?includeids?
Returns a list whose elements describe the visuals available for window's screen. Each element consists of a visual class followed by an integer depth. The class has the same form as returned by winfo visual. The depth gives the number of bits per pixel in the visual. In addition, if the includeids argument is provided, then the depth is followed by the X identifier for the visual.

winfo vrootheight window
Returns the height of the virtual root window associated with window if there is one; otherwise returns the height of window's screen.

winfo vrootwidth window
Returns the width of the virtual root window associated with window if there is one; otherwise returns the width of window's screen.

winfo vrootx window
Returns the x-offset of the virtual root window associated with window, relative to the root window of its screen. This is normally either zero or negative. Returns 0 if there is no virtual root window for window.

winfo vrooty window
Returns the y-offset of the virtual root window associated with window, relative to the root window of its screen. This is normally either zero or negative. Returns 0 if there is no virtual root window for window.

winfo width window
Returns a decimal string giving window's width in pixels. When a window is first created its width will be 1 pixel; the width will eventually be changed by a geometry manager to fulfill the
window's needs. If you need the true width immediately after creating a widget, invoke update to force the geometry manager to arrange it, or use winfo reqwidth to get the window's requested width instead of its actual width.

**winfo x window**
Returns a decimal string giving the x-coordinate, in window's parent, of the upper-left corner of window's border (or window if it has no border).

**winfo y window**
Returns a decimal string giving the y-coordinate, in window's parent, of the upper-left corner of window's border (or window if it has no border).

### EXAMPLE

Print where the mouse pointer is and what window it is currently over:

```tcl
lassign [winfo pointerxy .] x y
puts -nonewline "Mouse pointer at ($x,$y) which is "
set win [winfo containing $x $y]
if {$win eq ""} {
    puts "over no window"
} else {
    puts "over $win"
}
```

### KEYWORDS

atom, children, class, geometry, height, identifier, information, interpreters, mapped, parent, path name, screen, virtual root, width, window

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
NAME

`event` - Miscellaneous event facilities: define virtual events and generate events

SYNOPSIS

**DESCRIPTION**

```bash
event add <<virtual>> sequence ?sequence ...?
event delete <<virtual>> ?sequence sequence ...?
event generate window event ?option value option value ...?
event info ?<<virtual>>?
```

**EVENT FIELDS**

- `-above window`
- `-borderwidth size`
- `-button number`
- `-count number`
- `-data string`
- `-delta number`
- `-detail detail`
- `-focus boolean`
- `-height size`
- `-keycode number`
- `-keysym name`
- `-mode notify`
- `-override boolean`
- `-place where`
- `-root window`
- `-rootx coord`
- `-rooty coord`
- `-sendevent boolean`
- `-serial number`
- `-state state`
- `-subwindow window`

-time integer  
-warp boolean  
-width size  
-when when  
  now  
  tail  
  head  
  mark  
-x coord  
-y coord

PREDEFINED VIRTUAL EVENTS

<<AltUnderlined>>
<<ListboxSelect>>
<<MenuSelect>>
<<Modified>>
<<Selection>>
<<TraverseIn>>
<<TraverseOut>>
<<Clear>>
<<Copy>>
<<Cut>>
<<Paste>>
<<PasteSelection>>
<<PrevWindow>>
<<Redo>>
<<Undo>>

VIRTUAL EVENT EXAMPLES

SEE ALSO
KEYWORDS

<table>
<thead>
<tr>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>event - Miscellaneous event facilities: define virtual events and generate events</td>
</tr>
</tbody>
</table>

SYNOPSIS

```
event option ?arg arg ...?```
The **event** command provides several facilities for dealing with window system events, such as defining virtual events and synthesizing events. The command has several different forms, determined by the first argument. The following forms are currently supported:

**event add <<virtual>> sequence ?sequence ...?**
Associates the virtual event `virtual` with the physical event sequence(s) given by the `sequence` arguments, so that the virtual event will trigger whenever any one of the `sequences` occurs. `Virtual` may be any string value and `sequence` may have any of the values allowed for the `sequence` argument to the `bind` command. If `virtual` is already defined, the new physical event sequences add to the existing sequences for the event.

**event delete <<virtual>> ?sequence sequence ...?**
Deletes each of the `sequences` from those associated with the virtual event given by `virtual`. `Virtual` may be any string value and `sequence` may have any of the values allowed for the `sequence` argument to the `bind` command. Any `sequences` not currently associated with `virtual` are ignored. If no `sequence` argument is provided, all physical event sequences are removed for `virtual`, so that the virtual event will not trigger anymore.

**event generate window event ?option value option value ...?**
Generates a window event and arranges for it to be processed just as if it had come from the window system. `Window` gives the path name of the window for which the event will be generated; it may also be an identifier (such as returned by `winfo id`) as long as it is for a window in the current application. `Event` provides a basic description of the event, such as `<Shift-Button-2>` or `<<Paste>>`. If `Window` is empty the whole screen is meant, and coordinates are relative to the screen. `Event` may have any of the forms allowed for the `sequence` argument of the `bind` command except that it must consist of a single event pattern, not a sequence. `Option-value` pairs may be used to specify additional attributes of the event, such
as the x and y mouse position; see EVENT FIELDS below. If the -when option is not specified, the event is processed immediately: all of the handlers for the event will complete before the event generate command returns. If the -when option is specified then it determines when the event is processed. Certain events, such as key events, require that the window has focus to receive the event properly.

**event info ?<<<virtual>>??**

Returns information about virtual events. If the ?<<<virtual>>?> argument is omitted, the return value is a list of all the virtual events that are currently defined. If ?<<<virtual>>?> is specified then the return value is a list whose elements are the physical event sequences currently defined for the given virtual event; if the virtual event is not defined then an empty string is returned.

Note that virtual events that that are not bound to physical event sequences are not returned by event info.

## EVENT FIELDS

The following options are supported for the event generate command. These correspond to the “%” expansions allowed in binding scripts for the bind command.

**-above window**

*Window* specifies the above field for the event, either as a window path name or as an integer window id. Valid for Configure events. Corresponds to the %a substitution for binding scripts.

**-borderwidth size**

*Size* must be a screen distance; it specifies the border width field for the event. Valid for Configure events. Corresponds to the %B substitution for binding scripts.

**-button number**

*Number* must be an integer; it specifies the detail field for a ButtonPress or ButtonRelease event, overriding any button
number provided in the base event argument. Corresponds to the \%b substitution for binding scripts.

**-count number**

*Number* must be an integer; it specifies the *count* field for the event. Valid for **Expose** events. Corresponds to the \%c substitution for binding scripts.

**-data string**

*String* may be any value; it specifies the *user_data* field for the event. Only valid for virtual events. Corresponds to the \%d substitution for virtual events in binding scripts.

**-delta number**

*Number* must be an integer; it specifies the *delta* field for the **MouseWheel** event. The *delta* refers to the direction and magnitude the mouse wheel was rotated. Note the value is not a screen distance but are units of motion in the mouse wheel. Typically these values are multiples of 120. For example, 120 should scroll the text widget up 4 lines and -240 would scroll the text widget down 8 lines. Of course, other widgets may define different behaviors for mouse wheel motion. This field corresponds to the \%D substitution for binding scripts.

**-detail detail**

*Detail* specifies the *detail* field for the event and must be one of the following:

- NotifyAncestor
- NotifyNonlinearVirtual
- NotifyDetailNone
- NotifyPointer
- NotifyInferior
- NotifyPointerRoot
- NotifyNonlinear
- NotifyVirtual
Valid for **Enter, Leave, FocusIn** and **FocusOut** events. Corresponds to the %d substitution for binding scripts.

- **focus** *boolean*
  
  *Boolean* must be a boolean value; it specifies the *focus* field for the event. Valid for **Enter** and **Leave** events. Corresponds to the %f substitution for binding scripts.

- **height** *size*
  
  *Size* must be a screen distance; it specifies the *height* field for the event. Valid for **Configure** events. Corresponds to the %h substitution for binding scripts.

- **keycode** *number*
  
  *Number* must be an integer; it specifies the *keycode* field for the event. Valid for **KeyPress** and **KeyRelease** events. Corresponds to the %k substitution for binding scripts.

- **keysym** *name*
  
  *Name* must be the name of a valid keysym, such as *g, space, or Return*; its corresponding keycode value is used as the *keycode* field for event, overriding any detail specified in the base *event* argument. Valid for **KeyPress** and **KeyRelease** events. Corresponds to the %K substitution for binding scripts.

- **mode** *notify*
  
  *Notify* specifies the *mode* field for the event and must be one of **NotifyNormal, NotifyGrab, NotifyUngrab, or NotifyWhileGrabbed**. Valid for **Enter, Leave, FocusIn, and FocusOut** events. Corresponds to the %m substitution for binding scripts.

- **override** *boolean*
  
  *Boolean* must be a boolean value; it specifies the *override_redirect* field for the event. Valid for **Map, Reparent, and Configure** events. Corresponds to the %o substitution for binding scripts.

- **place** *where*
Where specifies the place field for the event; it must be either **PlaceOnTop** or **PlaceOnBottom**. Valid for **Circulate** events. Corresponds to the %p substitution for binding scripts.

**-root window**

*Window* must be either a window path name or an integer window identifier; it specifies the **root** field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, and **Motion** events. Corresponds to the %R substitution for binding scripts.

**-rootx coord**

*Coord* must be a screen distance; it specifies the x_root field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, and **Motion** events. Corresponds to the %X substitution for binding scripts.

**-rooty coord**

*Coord* must be a screen distance; it specifies the y_root field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, and **Motion** events. Corresponds to the %Y substitution for binding scripts.

**-sendevent** boolean

*Boolean* must be a boolean value; it specifies the send_event field for the event. Valid for all events. Corresponds to the %E substitution for binding scripts.

**-serial number**

*Number* must be an integer; it specifies the serial field for the event. Valid for all events. Corresponds to the %# substitution for binding scripts.

**-state state**

*State* specifies the state field for the event. For **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Enter**, **Leave**, and **Motion** events it must be an integer value. For **Visibility** events it must be one of **VisibilityUnobscured**,
VisibilityPartiallyObscured, or VisibilityFullyObscured. This option overrides any modifiers such as Meta or Control specified in the base event. Corresponds to the %s substitution for binding scripts.

-subwindow window
Window specifies the subwindow field for the event, either as a path name for a Tk widget or as an integer window identifier. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, and Motion events. Similar to %S substitution for binding scripts.

-time integer
Integer must be an integer value; it specifies the time field for the event. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, Enter, Leave, Motion, and Property events. Corresponds to the %t substitution for binding scripts.

-warp boolean
boolean must be a boolean value; it specifies whether the screen pointer should be warped as well. Valid for KeyPress, KeyRelease, ButtonPress, ButtonRelease, and Motion events. The pointer will only warp to a window if it is mapped.

-width size
Size must be a screen distance; it specifies the width field for the event. Valid for Configure events. Corresponds to the %w substitution for binding scripts.

-when when
When determines when the event will be processed; it must have one of the following values:

now
Process the event immediately, before the command returns. This also happens if the -when option is omitted.

tail
Place the event on Tcl's event queue behind any events already queued for this application.

**head**
Place the event at the front of Tcl's event queue, so that it will be handled before any other events already queued.

**mark**
Place the event at the front of Tcl's event queue but behind any other events already queued with **-when mark**. This option is useful when generating a series of events that should be processed in order but at the front of the queue.

**-x coord**
Coord must be a screen distance; it specifies the x field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Motion**, **Enter**, **Leave**, **Expose**, **Configure**, **Gravity**, and **Reparent** events. Corresponds to the %x substitution for binding scripts. If **Window** is empty the coordinate is relative to the screen, and this option corresponds to the %X substitution for binding scripts.

**-y coord**
Coord must be a screen distance; it specifies the y field for the event. Valid for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, **Motion**, **Enter**, **Leave**, **Expose**, **Configure**, **Gravity**, and **Reparent** events. Corresponds to the %y substitution for binding scripts. If **Window** is empty the coordinate is relative to the screen, and this option corresponds to the %Y substitution for binding scripts.

Any options that are not specified when generating an event are filled with the value 0, except for **serial**, which is filled with the next X event serial number.

---

**PREDEFINED VIRTUAL EVENTS**

Tk defines the following virtual events for the purposes of notification:
This is sent to widget to notify it that the letter it has underlined (as an accelerator indicator) with the \texttt{-underline} option has been pressed in combination with the Alt key. The usual response to this is to either focus into the widget (or some related widget) or to invoke the widget.

This is sent to a listbox when the set of selected item(s) in the listbox is updated.

This is sent to a menu when the currently selected item in the menu changes. It is intended for use with context-sensitive help systems.

This is sent to a text widget when the contents of the widget are changed.

This is sent to a text widget when the selection in the widget is changed.

This is sent to a widget when the focus enters the widget because of a user-driven “tab to widget” action.

This is sent to a widget when the focus leaves the widget because of a user-driven “tab to widget” action.

Tk defines the following virtual events for the purposes of unifying bindings across multiple platforms. Users expect them to behave in the following way:

Delete the currently selected widget contents.
Copy the currently selected widget contents to the clipboard.

<<Cut>>
Move the currently selected widget contents to the clipboard.

<<Paste>>
Replace the currently selected widget contents with the contents of the clipboard.

<<PasteSelection>>
Insert the contents of the selection at the mouse location. (This event has meaningful \%x and \%y substitutions).

<<PrevWindow>>
Traverse to the previous window.

<<Redo>>
Redo one undone action.

<<Undo>>
Undo the last action.

VIRTUAL EVENT EXAMPLES

In order for a virtual event binding to trigger, two things must happen. First, the virtual event must be defined with the \texttt{event add} command. Second, a binding must be created for the virtual event with the \texttt{bind} command. Consider the following virtual event definitions:

\begin{verbatim}
  event add <<Paste>> <Control-y>
  event add <<Paste>> <Button-2>
  event add <<Save>> <Control-X><Control-S>
  event add <<Save>> <Shift-F12>
\end{verbatim}

In the \texttt{bind} command, a virtual event can be bound like any other builtin event type as follows:
bind Entry <<Paste>> {\%W insert [selection get]}

The double angle brackets are used to specify that a virtual event is being bound. If the user types Control-y or presses button 2, or if a <<Paste>> virtual event is synthesized with `event generate`, then the <<Paste>> binding will be invoked.

If a virtual binding has the exact same sequence as a separate physical binding, then the physical binding will take precedence. Consider the following example:

```
bind Entry <Control-y> {puts Control-y}
bind Entry <<Paste>> {puts Paste}
```

When the user types Control-y the `<Control-y>` binding will be invoked, because a physical event is considered more specific than a virtual event, all other things being equal. However, when the user types Meta-Control-y the <<Paste>> binding will be invoked, because the Meta modifier in the physical pattern associated with the virtual binding is more specific than the `<Control-y>` sequence for the physical event.

Bindings on a virtual event may be created before the virtual event exists. Indeed, the virtual event never actually needs to be defined, for instance, on platforms where the specific virtual event would meaningless or ungeneratable.

When a definition of a virtual event changes at run time, all windows will respond immediately to the new definition. Starting from the preceding example, if the following code is executed:

```
bind <Entry> <Control-y> {}
event add <<Paste>> <Key-F6>
```
the behavior will change such in two ways. First, the shadowed `<<Paste>>` binding will emerge. Typing Control-y will no longer invoke the `<Control-y>` binding, but instead invoke the virtual event `<<Paste>>`. Second, pressing the F6 key will now also invoke the `<<Paste>>` binding.

SEE ALSO

bind

KEYWORDS

define, handle, virtual event
NAME
message - Create and manipulate message widgets

SYNOPSIS

STANDARD OPTIONS
-anchor, anchor, Anchor
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable

WIDGET-SPECIFIC OPTIONS
-aspect, aspect, Aspect
-justify, justify, Justify
-width, width, Width

DESCRIPTION

WIDGET COMMAND

pathName cget option

pathName configure ?option? ?value option value ...?

DEFAULT BINDINGS

BUGS

SEE ALSO
KEYWORDS

NAME

message - Create and manipulate message widgets

SYNOPSIS

message pathName ?options?

STANDARD OPTIONS

-anchor, anchor, Anchor
-background or -bg, background, Background
-borderwidth or -bd, borderWidth, BorderWidth
-cursor, cursor, Cursor
-font, font, Font
-foreground or -fg, foreground, Foreground
-highlightbackground, highlightBackground, HighlightBackground
-highlightcolor, highlightColor, HighlightColor
-highlightthickness, highlightThickness, HighlightThickness
-padx, padX, Pad
-pady, padY, Pad
-relief, relief, Relief
-takefocus, takeFocus, TakeFocus
-text, text, Text
-textvariable, textVariable, Variable

WIDGET-SPECIFIC OPTIONS

Command-Line Name: -aspect
Database Name: aspect
Database Class: Aspect

Specifies a non-negative integer value indicating desired aspect ratio for the text. The aspect ratio is specified as 100*width/height. 100 means the text should be as wide as it is tall, 200 means the text should be twice as wide as it is tall, 50 means the text should be twice as tall as it is wide, and so on. Used to choose line length
Specifies how to justify lines of text. Must be one of left, center, or right. Defaults to left. This option works together with the anchor, aspect, padX, padY, and width options to provide a variety of arrangements of the text within the window. The aspect and width options determine the amount of screen space needed to display the text. The anchor, padX, and padY options determine where this rectangular area is displayed within the widget's window, and the justify option determines how each line is displayed within that rectangular region. For example, suppose anchor is e and justify is left, and that the message window is much larger than needed for the text. The text will be displayed so that the left edges of all the lines line up and the right edge of the longest line is padX from the right side of the window; the entire text block will be centered in the vertical span of the window.

Specifies the length of lines in the window. The value may have any of the forms acceptable to Tk_GetPixels. If this option has a value greater than zero then the aspect option is ignored and the width option determines the line length. If this option has a value less than or equal to zero, then the aspect option determines the line length.

The message command creates a new window (given by the pathName argument) and makes it into a message widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the message such as its colors, font, text, and initial relief. The message command returns its pathName argument. At the time this command is invoked, there must...
not exist a window named *pathName*, but *pathName*'s parent must exist.

A message is a widget that displays a textual string. A message widget has three special features. First, it breaks up its string into lines in order to produce a given aspect ratio for the window. The line breaks are chosen at word boundaries wherever possible (if not even a single word would fit on a line, then the word will be split across lines). Newline characters in the string will force line breaks; they can be used, for example, to leave blank lines in the display.

The second feature of a message widget is justification. The text may be displayed left-justified (each line starts at the left side of the window), centered on a line-by-line basis, or right-justified (each line ends at the right side of the window).

The third feature of a message widget is that it handles control characters and non-printing characters specially. Tab characters are replaced with enough blank space to line up on the next 8-character boundary. Newlines cause line breaks. Other control characters (ASCII code less than 0x20) and characters not defined in the font are displayed as a four-character sequence `\xhh` where `hh` is the two-digit hexadecimal number corresponding to the character. In the unusual case where the font does not contain all of the characters in "0123456789abcdef\x" then control characters and undefined characters are not displayed at all.

**WIDGET COMMAND**

The **message** command creates a new Tcl command whose name is *pathName*. This command may be used to invoke various operations on the widget. It has the following general form:

```
    pathName option ?arg arg ...?
```

*Option* and the *args* determine the exact behavior of the command. The
following commands are possible for message widgets:

*pathName cget option*
    Returns the current value of the configuration option given by
    *option*. *Option* may have any of the values accepted by the
    *message* command.

*pathName configure ?option? ?value option value ...?*
    Query or modify the configuration options of the widget. If no *option*
    is specified, returns a list describing all of the available options for
    *pathName* (see Tk_ConfigureInfo for information on the format of
    this list). If *option* is specified with no *value*, then the command
    returns a list describing the one named option (this list will be
    identical to the corresponding sublist of the value returned if no
    *option* is specified). If one or more *option-value* pairs are specified,
    then the command modifies the given widget option(s) to have the
    given value(s); in this case the command returns an empty string.
    *Option* may have any of the values accepted by the *message*
    command.

**DEFAULT BINDINGS**

When a new message is created, it has no default event bindings:
messages are intended for output purposes only.

**BUGS**

Tabs do not work very well with text that is centered or right-justified.
The most common result is that the line is justified wrong.

**SEE ALSO**

label

**KEYWORDS**

message, widget
NAME

tk - Manipulate Tk internal state

SYNOPSIS

DESCRIPTION

The tk command provides access to miscellaneous elements of Tk's internal state. Most of the information manipulated by this command pertains to the application as a whole, or to a screen or display, rather than to a particular window. The command can take any of a number of different forms depending on the option argument. The legal forms are:

**tk appname ?newName?**

If *newName* is not specified, this command returns the name of the application (the name that may be used in send commands to
communicate with the application). If newName is specified, then the name of the application is changed to newName. If the given name is already in use, then a suffix of the form “#2” or “#3” is appended in order to make the name unique. The command’s result is the name actually chosen. newName should not start with a capital letter. This will interfere with option processing, since names starting with capitals are assumed to be classes; as a result, Tk may not be able to find some options for the application. If sends have been disabled by deleting the send command, this command will reenable them and recreate the send command.

**tk caret window** ?-x x? ?-y y? ?-height height?
Sets and queries the caret location for the display of the specified Tk window window. The caret is the per-display cursor location used for indicating global focus (e.g. to comply with Microsoft Accessibility guidelines), as well as for location of the over-the-spot XIM (X Input Methods) or Windows IME windows. If no options are specified, the last values used for setting the caret are return in option-value pair format. -x and -y represent window-relative coordinates, and -height is the height of the current cursor location, or the height of the specified window if none is given.

**tk scaling** ?-displayof window? ?number?
Sets and queries the current scaling factor used by Tk to convert between physical units (for example, points, inches, or millimeters) and pixels. The number argument is a floating point number that specifies the number of pixels per point on window's display. If the window argument is omitted, it defaults to the main window. If the number argument is omitted, the current value of the scaling factor is returned.

A “point” is a unit of measurement equal to 1/72 inch. A scaling factor of 1.0 corresponds to 1 pixel per point, which is equivalent to a standard 72 dpi monitor. A scaling factor of 1.25 would mean 1.25 pixels per point, which is the setting for a 90 dpi monitor; setting the scaling factor to 1.25 on a 72 dpi monitor would cause everything in the application to be displayed 1.25 times as large as normal. The initial value for the scaling factor is set when the application starts,
based on properties of the installed monitor, but it can be changed at any time. Measurements made after the scaling factor is changed will use the new scaling factor, but it is undefined whether existing widgets will resize themselves dynamically to accommodate the new scaling factor.

**tk inactive ?-displayof window? ?reset?**
Returns a positive integer, the number of milliseconds since the last time the user interacted with the system. If the `-displayof` option is given then the return value refers to the display of `window`; otherwise it refers to the display of the application's main window.

`tk inactive` will return -1, if querying the user inactive time is not supported by the system, and in safe interpreters.

If the literal string `reset` is given as an additional argument, the timer is reset and an empty string is returned. Resetting the inactivity time is forbidden in safe interpreters and will throw and error if tried.

**tk useinputmethods ?-displayof window? ?boolean?**
Sets and queries the state of whether Tk should use XIM (X Input Methods) for filtering events. The resulting state is returned. XIM is used in some locales (i.e., Japanese, Korean), to handle special input devices. This feature is only significant on X. If XIM support is not available, this will always return 0. If the `window` argument is omitted, it defaults to the main window. If the `boolean` argument is omitted, the current state is returned. This is turned on by default for the main display.

**tk windowingsystem**
Returns the current Tk windowing system, one of **x11** (X11-based), **win32** (MS Windows), or **aqua** (Mac OS X Aqua).

**SEE ALSO**

send, winfo
NAME
   ttk::menubutton - Widget that pops down a menu when pressed

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS
   -class
   -compound, compound, Compound
   -cursor, cursor, Cursor
   -image, image, Image
   -state
   -style
   -takefocus, takeFocus, TakeFocus
   -text, text, Text
   -textvariable, textVariable, Variable
   -underline, underline, Underline
   -width

WIDGET-SPECIFIC OPTIONS
   -direction, direction, Direction
   -menu, menu, Menu

WIDGET COMMAND

SEE ALSO

KEYWORDS

NAME

   ttk::menubutton - Widget that pops down a menu when pressed

SYNOPSIS

   ttk::menubutton pathName ?options?
A `ttk::menubutton` widget displays a textual label and/or image, and displays a menu when pressed.

**STANDARD OPTIONS**

- `class`
- `compound`, `compound`, `Compound`
- `cursor`, `cursor`, `Cursor`
- `image`, `image`, `Image`
- `state`
- `style`
- `takefocus`, `takeFocus`, `TakeFocus`
- `text`, `text`, `Text`
- `textvariable`, `textVariable`, `Variable`
- `underline`, `underline`, `Underline`
- `width`

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: `-direction`
Database Name: `direction`
Database Class: `Direction`
  Specifies where the menu is to be popped up relative to the menubutton. One of: `above`, `below`, `left`, `right`, or `flush`. The default is `below`. `flush` pops the menu up directly over the menubutton.

Command-Line Name: `-menu`
Database Name: `menu`
Database Class: `Menu`
  Specifies the path name of the menu associated with the menubutton. To be on the safe side, the menu ought to be a direct child of the menubutton.
Menubutton widgets support the standard `cget`, `configure`, `identify`, `instate`, and `state` methods. No other widget methods are used.

**SEE ALSO**

`ttk::widget`, `menu`, `menubutton`

**KEYWORDS**

`widget`, `button`, `menu`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2004 Joe English
NAME
wm - Communicate with window manager

SYNOPSIS

DESCRIPTION

wm aspect window ?minNumer minDenom maxNumer maxDenom?
wm attributes window
wm attributes window ?option?
wm attributes window ?option value option value...?
  -fullscreen
  -topmost
  -alpha
  -disabled
  -toolwindow
  -transparentcolor
  -alpha
  -modified
  -notify
  -titlepath
  -transparent
  -zoomed
wm client window ?name?
wm colormapwindows window ?windowList?
wm command window ?value?
wm deiconify window
wm focusmodel window ?active|passive?
wm forget window
wm frame window
wm geometry window ?newGeometry?
wm grid window ?baseWidth baseHeight widthInc heightInc?
wm group window ?pathName?
wm iconbitmap window ?bitmap?  
  wm iconbitmap window ?-default? ?image?
wm iconify window
wm iconmask window ?bitmap?
wm iconname window ?newName?
wm iconphoto window ?-default? image1 ?image2 ...?
wm iconposition window ?x y?
wm iconwindow window ?pathName?
wm manage widget
wm maxsize window ?width height?
wm minsize window ?width height?
wm overrideredirect window ?boolean?
wm positionfrom window ?who?
wm protocol window ?name? ?command?
wm resizable window ?width height?
wm sizefrom window ?who?
wm stackorder window ?isabove|isbelow window?
wm state window ?newstate?
wm title window ?string?
wm transient window ?master?
wm withdraw window

GEOMETRY MANAGEMENT  
GRIDDED GEOMETRY MANAGEMENT  
BUGS  
EXAMPLES  
SEE ALSO  
KEYWORDS

**NAME**

wm - Communicate with window manager

**SYNOPSIS**

wm option window ?args?

**DESCRIPTION**
The `wm` command is used to interact with window managers in order to control such things as the title for a window, its geometry, or the increments in terms of which it may be resized. The `wm` command can take any of a number of different forms, depending on the `option` argument. All of the forms expect at least one additional argument, `window`, which must be the path name of a top-level window.

The legal forms for the `wm` command are:

**wm aspect** window ?minNumer minDenom maxNumer maxDenom?

If `minNumer`, `minDenom`, `maxNumer`, and `maxDenom` are all specified, then they will be passed to the window manager and the window manager should use them to enforce a range of acceptable aspect ratios for `window`. The aspect ratio of `window` (width/length) will be constrained to lie between `minNumer/minDenom` and `maxNumer/maxDenom`. If `minNumer` etc. are all specified as empty strings, then any existing aspect ratio restrictions are removed. If `minNumer` etc. are specified, then the command returns an empty string. Otherwise, it returns a Tcl list containing four elements, which are the current values of `minNumer`, `minDenom`, `maxNumer`, and `maxDenom` (if no aspect restrictions are in effect, then an empty string is returned).

**wm attributes** window

**wm attributes** window ?option?

**wm attributes** window ?option value option value...?

This subcommand returns or sets platform specific attributes associated with a window. The first form returns a list of the platform specific flags and their values. The second form returns the value for the specific option. The third form sets one or more of the values. The values are as follows:

All platforms support the following attributes (though X11 users should see the notes below):

- `fullscreen`
Places the window in a mode that takes up the entire screen, has no borders, and covers the general use area (i.e. Start menu and taskbar on Windows, dock and menubar on OSX, general window decorations on X11).

-topmost
Specifies whether this is a topmost window (displays above all other windows).

On Windows, the following attributes may be set.

-alpha
Specifies the alpha transparency level of the toplevel. It accepts a value from 0.0 (fully transparent) to 1.0 (opaque). Values outside that range will be constrained. This is supported on Windows 2000/XP+. Where not supported, the -alpha value remains at 1.0.

-disabled
Specifies whether the window is in a disabled state.

-toolwindow
Specifies a toolwindow style window (as defined in the MSDN).

-transparentcolor
Specifies the transparent color index of the toplevel. It takes any color value accepted by Tk_GetColor. If the empty string is specified (default), no transparent color is used. This is supported on Windows 2000/XP+. Where not supported, the -transparentcolor value remains at {}.

On Mac OS X, the following attributes may be set.

-alpha
Specifies the alpha transparency level of the window. It accepts a value from 0.0 (fully transparent) to 1.0 (opaque), values outside that range will be constrained.

-modified
Specifies the modification state of the window (determines whether the window close widget contains the modification indicator and whether the proxy icon is draggable).

- **notify**
  Specifies process notification state (bouncing of the application dock icon).

- **titlepath**
  Specifies the path of the file referenced as the window proxy icon (which can be dragged and dropped in lieu of the file's finder icon).

- **transparent**
  Makes the window content area transparent and turns off the window shadow. For the transparency to be effective, the toplevel background needs to be set to a color with some alpha, e.g. “systemTransparent”.

On X11, the following attributes may be set. These are not supported by all window managers, and will have no effect under older WMs.

- **zoomed**
  Requests that the window should be maximized. This is the same as `wm state zoomed` on Windows and Mac OS X.

On X11, changes to window attributes are performed asynchronously. Querying the value of an attribute returns the current state, which will not be the same as the value most recently set if the window manager has not yet processed the request or if it does not support the attribute.

**wm client window ?name?**
If `name` is specified, this command stores `name` (which should be the name of the host on which the application is executing) in `window`'s **WM_CLIENT_MACHINE** property for use by the window manager or session manager. The command returns an empty
string in this case. If *name* is not specified, the command returns the last name set in a `wm client` command for *window*. If *name* is specified as an empty string, the command deletes the `WM_CLIENT_MACHINE` property from *window*.

`wm colormapwindows window ?windowList?`  
This command is used to manipulate the `WM_COLORMAP_WINDOWS` property, which provides information to the window managers about windows that have private colormaps.

If *windowList* is not specified, the command returns a list whose elements are the names of the windows in the `WM_COLORMAP_WINDOWS` property. If *windowList* is specified, it consists of a list of window path names; the command overwrites the `WM_COLORMAP_WINDOWS` property with the given windows and returns an empty string. The `WM_COLORMAP_WINDOWS` property should normally contain a list of the internal windows within *window* whose colormaps differ from their parents.

The order of the windows in the property indicates a priority order: the window manager will attempt to install as many colormaps as possible from the head of this list when *window* gets the colormap focus. If *window* is not included among the windows in *windowList*, Tk implicitly adds it at the end of the `WM_COLORMAP_WINDOWS` property, so that its colormap is lowest in priority. If `wm colormapwindows` is not invoked, Tk will automatically set the property for each top-level window to all the internal windows whose colormaps differ from their parents, followed by the top-level itself; the order of the internal windows is undefined. See the ICCCM documentation for more information on the `WM_COLORMAP_WINDOWS` property.

`wm command window ?value?`  
If *value* is specified, this command stores *value* in *window*'s `WM_COMMAND` property for use by the window manager or session manager and returns an empty string. *Value* must have proper list structure; the elements should contain the words of the
command used to invoke the application. If value is not specified then the command returns the last value set in a **wm command** command for *window*. If value is specified as an empty string, the command deletes the **WM_COMMAND** property from *window*.

**wm deiconify** *window*
Arrange for *window* to be displayed in normal (non-iconified) form. This is done by mapping the window. If the window has never been mapped then this command will not map the window, but it will ensure that when the window is first mapped it will be displayed in de-iconified form. On Windows, a deiconified window will also be raised and be given the focus (made the active window). Returns an empty string.

**wm focusmodel** *window* ?active|passive?
If active or passive is supplied as an optional argument to the command, then it specifies the focus model for *window*. In this case the command returns an empty string. If no additional argument is supplied, then the command returns the current focus model for *window*.

An **active** focus model means that *window* will claim the input focus for itself or its descendants, even at times when the focus is currently in some other application. **Passive** means that *window* will never claim the focus for itself: the window manager should give the focus to *window* at appropriate times. However, once the focus has been given to *window* or one of its descendants, the application may re-assign the focus among *window's* descendants. The focus model defaults to **passive**, and Tk's **focus** command assumes a passive model of focusing.

**wm forget** *window*
The *window* will be unmapped from the screen and will no longer be managed by **wm**. Windows created with the **toplevel** command will be treated like **frame** windows once they are no longer managed by **wm**, however, the **-menu** configuration will be remembered and the menus will return once the widget is managed again.
**wm frame** *window*

If *window* has been reparented by the window manager into a decorative frame, the command returns the platform specific window identifier for the outermost frame that contains *window* (the window whose parent is the root or virtual root). If *window* has not been reparented by the window manager then the command returns the platform specific window identifier for *window*.

**wm geometry** *window* ?*newGeometry*?

If *newGeometry* is specified, then the geometry of *window* is changed and an empty string is returned. Otherwise the current geometry for *window* is returned (this is the most recent geometry specified either by manual resizing or in a **wm geometry** command). *NewGeometry* has the form =*width*×*height*±*x*±*y*, where any of =, *width*×*height*, or ±*x*±*y* may be omitted. *Width* and *height* are positive integers specifying the desired dimensions of *window*. If *window* is gridded (see **GRIDDED GEOMETRY MANAGEMENT** below) then the dimensions are specified in grid units; otherwise they are specified in pixel units.

*X* and *y* specify the desired location of *window* on the screen, in pixels. If *x* is preceded by +, it specifies the number of pixels between the left edge of the screen and the left edge of *window*’s border; if preceded by - then *x* specifies the number of pixels between the right edge of the screen and the right edge of *window*’s border. If *y* is preceded by + then it specifies the number of pixels between the top of the screen and the top of *window*’s border; if *y* is preceded by - then it specifies the number of pixels between the bottom of *window*’s border and the bottom of the screen.

If *newGeometry* is specified as an empty string then any existing user-specified geometry for *window* is cancelled, and the window will revert to the size requested internally by its widgets.

**wm grid** *window* ?*baseWidth* *baseHeight* *widthInc* *heightInc*?

This command indicates that *window* is to be managed as a gridded window. It also specifies the relationship between grid units
and pixel units. *BaseWidth* and *baseHeight* specify the number of grid units corresponding to the pixel dimensions requested internally by *window* using *Tk_GeometryRequest*. *WidthInc* and *heightInc* specify the number of pixels in each horizontal and vertical grid unit. These four values determine a range of acceptable sizes for *window*, corresponding to grid-based widths and heights that are non-negative integers. Tk will pass this information to the window manager; during manual resizing, the window manager will restrict the window's size to one of these acceptable sizes.

Furthermore, during manual resizing the window manager will display the window's current size in terms of grid units rather than pixels. If *baseWidth* etc. are all specified as empty strings, then *window* will no longer be managed as a gridded window. If *baseWidth* etc. are specified then the return value is an empty string.

Otherwise the return value is a Tcl list containing four elements corresponding to the current *baseWidth*, *baseHeight*, *widthInc*, and *heightInc*; if *window* is not currently gridded, then an empty string is returned.

Note: this command should not be needed very often, since the *Tk_SetGrid* library procedure and the *setGrid* option provide easier access to the same functionality.

**wm group** *window* *?pathName?*

If *pathName* is specified, it gives the path name for the leader of a group of related windows. The window manager may use this information, for example, to unmap all of the windows in a group when the group's leader is iconified. *PathName* may be specified as an empty string to remove *window* from any group association. If *pathName* is specified then the command returns an empty string; otherwise it returns the path name of *window*'s current group leader, or an empty string if *window* is not part of any group.

**wm iconbitmap** *window* *?bitmap?*
If *bitmap* is specified, then it names a bitmap in the standard forms accepted by Tk (see the [Tk_GetBitmap](#) manual entry for details). This bitmap is passed to the window manager to be displayed in *window*'s icon, and the command returns an empty string. If an empty string is specified for *bitmap*, then any current icon bitmap is cancelled for *window*. If *bitmap* is specified then the command returns an empty string. Otherwise it returns the name of the current icon bitmap associated with *window*, or an empty string if *window* has no icon bitmap. On the Windows operating system, an additional flag is supported:

**wm iconbitmap window ?-default? ?image?**

If the **-default** flag is given, the icon is applied to all toplevel windows (existing and future) to which no other specific icon has yet been applied. In addition to bitmap image types, a full path specification to any file which contains a valid Windows icon is also accepted (usually .ico or .icr files), or any file for which the shell has assigned an icon. Tcl will first test if the file contains an icon, then if it has an assigned icon, and finally, if that fails, test for a bitmap.

**wm iconify window**

Arrange for *window* to be iconified. It *window* has not yet been mapped for the first time, this command will arrange for it to appear in the iconified state when it is eventually mapped.

**wm iconmask window ?bitmap?**

If *bitmap* is specified, then it names a bitmap in the standard forms accepted by Tk (see the [Tk_GetBitmap](#) manual entry for details). This bitmap is passed to the window manager to be used as a mask in conjunction with the **iconbitmap** option: where the mask has zeroes no icon will be displayed; where it has ones, the bits from the icon bitmap will be displayed. If an empty string is specified for *bitmap* then any current icon mask is cancelled for *window* (this is equivalent to specifying a bitmap of all ones). If *bitmap* is specified then the command returns an empty string. Otherwise it returns the name of the current icon mask associated with *window*, or an empty string if no mask is in effect.
**wm iconname** *window* ?*newName*?

If *newName* is specified, then it is passed to the window manager; the window manager should display *newName* inside the icon associated with *window*. In this case an empty string is returned as result. If *newName* is not specified then the command returns the current icon name for *window*, or an empty string if no icon name has been specified (in this case the window manager will normally display the window's title, as specified with the **wm title** command).

**wm iconphoto** *window* ?*-default*? *image1* ?*image2* ...?

Sets the titlebar icon for *window* based on the named photo images. If **-default** is specified, this is applied to all future created toplevels as well. The data in the images is taken as a snapshot at the time of invocation. If the images are later changed, this is not reflected to the titlebar icons. Multiple images are accepted to allow different images sizes (e.g., 16x16 and 32x32) to be provided. The window manager may scale provided icons to an appropriate size.

On Windows, the images are packed into a Windows icon structure. This will override an ico specified to **wm iconbitmap**, and vice versa.

On X, the images are arranged into the _NET_WM_ICON X property, which most modern window managers support. A **wm iconbitmap** may exist simultaneously. It is recommended to use not more than 2 icons, placing the larger icon first.

On Macintosh, this currently does nothing.

**wm iconposition** *window* ?*x* *y*?

If *x* and *y* are specified, they are passed to the window manager as a hint about where to position the icon for *window*. In this case an empty string is returned. If *x* and *y* are specified as empty strings then any existing icon position hint is cancelled. If neither *x* nor *y* is specified, then the command returns a Tcl list containing two values, which are the current icon position hints (if no hints are in effect then an empty string is returned).
wm iconwindow window ?pathName?  
If pathName is specified, it is the path name for a window to use as icon for window: when window is iconified then pathName will be mapped to serve as icon, and when window is de-iconified then pathName will be unmapped again. If pathName is specified as an empty string then any existing icon window association for window will be cancelled. If the pathName argument is specified then an empty string is returned. Otherwise the command returns the path name of the current icon window for window, or an empty string if there is no icon window currently specified for window. Button press events are disabled for window as long as it is an icon window; this is needed in order to allow window managers to “own” those events. Note: not all window managers support the notion of an icon window.

wm manage widget  
The widget specified will become a stand alone top-level window. The window will be decorated with the window managers title bar, etc. Only frame, labelframe and toplevel widgets can be used with this command. Attempting to pass any other widget type will raise an error. Attempting to manage a toplevel widget is benign and achieves nothing. See also GEOMETRY MANAGEMENT.

wm maxsize window ?width height?  
If width and height are specified, they give the maximum permissible dimensions for window. For gridded windows the dimensions are specified in grid units; otherwise they are specified in pixel units. The window manager will restrict the window's dimensions to be less than or equal to width and height. If width and height are specified, then the command returns an empty string. Otherwise it returns a Tcl list with two elements, which are the maximum width and height currently in effect. The maximum size defaults to the size of the screen. See the sections on geometry management below for more information.

wm minsize window ?width height?  
If width and height are specified, they give the minimum permissible dimensions for window. For gridded windows the
dimensions are specified in grid units; otherwise they are specified in pixel units. The window manager will restrict the window’s dimensions to be greater than or equal to \textit{width} and \textit{height}. If \textit{width} and \textit{height} are specified, then the command returns an empty string. Otherwise it returns a Tcl list with two elements, which are the minimum width and height currently in effect. The minimum size defaults to one pixel in each dimension. See the sections on geometry management below for more information.

\textbf{wm overrideredirect window \textit{?boolean}?}

If \textit{boolean} is specified, it must have a proper boolean form and the override-redirect flag for \textit{window} is set to that value. If \textit{boolean} is not specified then 1 or 0 is returned to indicate whether or not the override-redirect flag is currently set for \textit{window}. Setting the override-redirect flag for a window causes it to be ignored by the window manager; among other things, this means that the window will not be reparented from the root window into a decorative frame and the user will not be able to manipulate the window using the normal window manager mechanisms.

\textbf{wm positionfrom window \textit{?who}?}

If \textit{who} is specified, it must be either \texttt{program} or \texttt{user}, or an abbreviation of one of these two. It indicates whether \textit{window}’s current position was requested by the program or by the user. Many window managers ignore program-requested initial positions and ask the user to manually position the window; if \texttt{user} is specified then the window manager should position the window at the given place without asking the user for assistance. If \textit{who} is specified as an empty string, then the current position source is cancelled. If \textit{who} is specified, then the command returns an empty string. Otherwise it returns \texttt{user} or \texttt{program} to indicate the source of the window’s current position, or an empty string if no source has been specified yet. Most window managers interpret “no source” as equivalent to \texttt{program}. Tk will automatically set the position source to \texttt{user} when a \texttt{wm geometry} command is invoked, unless the source has been set explicitly to \texttt{program}.

\textbf{wm protocol window \textit{?name}? \textit{?command}?}
This command is used to manage window manager protocols such as `WM_DELETE_WINDOW`. `Name` is the name of an atom corresponding to a window manager protocol, such as `WM_DELETE_WINDOW` or `WM_SAVE_YOURSELF` or `WM_TAKE_FOCUS`. If both `name` and `command` are specified, then `command` is associated with the protocol specified by `name`. `Name` will be added to `window`'s `WM_PROTOCOLS` property to tell the window manager that the application has a protocol handler for `name`, and `command` will be invoked in the future whenever the window manager sends a message to the client for that protocol. In this case the command returns an empty string. If `name` is specified but `command` is not, then the current command for `name` is returned, or an empty string if there is no handler defined for `name`. If `command` is specified as an empty string then the current handler for `name` is deleted and it is removed from the `WM_PROTOCOLS` property on `window`; an empty string is returned. Lastly, if neither `name` nor `command` is specified, the command returns a list of all the protocols for which handlers are currently defined for `window`.

Tk always defines a protocol handler for `WM_DELETE_WINDOW`, even if you have not asked for one with `wm protocol`. If a `WM_DELETE_WINDOW` message arrives when you have not defined a handler, then Tk handles the message by destroying the window for which it was received.

`wm resizable window ?width height?`
This command controls whether or not the user may interactively resize a top-level window. If `width` and `height` are specified, they are boolean values that determine whether the width and height of `window` may be modified by the user. In this case the command returns an empty string. If `width` and `height` are omitted then the command returns a list with two 0/1 elements that indicate whether the width and height of `window` are currently resizable. By default, windows are resizable in both dimensions. If resizing is disabled, then the window's size will be the size from the most recent interactive resize or `wm geometry` command. If there has been no such operation then the window's natural size will be used.
**wm sizefrom window ?who?**

If *who* is specified, it must be either **program** or **user**, or an abbreviation of one of these two. It indicates whether *window*'s current size was requested by the program or by the user. Some window managers ignore program-requested sizes and ask the user to manually size the window; if **user** is specified then the window manager should give the window its specified size without asking the user for assistance. If *who* is specified as an empty string, then the current size source is cancelled. If *who* is specified, then the command returns an empty string. Otherwise it returns **user** or **window** to indicate the source of the window's current size, or an empty string if no source has been specified yet. Most window managers interpret “no source” as equivalent to **program**.

**wm stackorder window ?isabove|isbelow window?**

The **stackorder** command returns a list of toplevel windows in stacking order, from lowest to highest. When a single toplevel window is passed, the returned list recursively includes all of the window's children that are toplevels. Only those toplevels that are currently mapped to the screen are returned. The **stackorder** command can also be used to determine if one toplevel is positioned above or below a second toplevel. When two window arguments separated by either **isabove** or **isbelow** are passed, a boolean result indicates whether or not the first window is currently above or below the second window in the stacking order.

**wm state window ?newstate?**

If *newstate* is specified, the window will be set to the new state, otherwise it returns the current state of *window*: either **normal**, **iconic**, **withdrawn**, **icon**, or (Windows and Mac OS X only) **zoomed**. The difference between **iconic** and **icon** is that **iconic** refers to a window that has been iconified (e.g., with the **wm iconify** command) while **icon** refers to a window whose only purpose is to serve as the icon for some other window (via the **wm iconwindow** command). The **icon** state cannot be set.

**wm title window ?string?**

If *string* is specified, then it will be passed to the window manager
for use as the title for `window` (the window manager should display this string in `window`'s title bar). In this case the command returns an empty string. If `string` is not specified then the command returns the current title for the `window`. The title for a window defaults to its name.

**wm transient window ?master?**

If `master` is specified, then the window manager is informed that `window` is a transient window (e.g. pull-down menu) working on behalf of `master` (where `master` is the path name for a top-level window). If `master` is specified as an empty string then `window` is marked as not being a transient window any more. Otherwise the command returns the path name of `window`'s current master, or an empty string if `window` is not currently a transient window. A transient window will mirror state changes in the master and inherit the state of the master when initially mapped. It is an error to attempt to make a window a transient of itself.

**wm withdraw window**

Arranges for `window` to be withdrawn from the screen. This causes the window to be unmapped and forgotten about by the window manager. If the window has never been mapped, then this command causes the window to be mapped in the withdrawn state. Not all window managers appear to know how to handle windows that are mapped in the withdrawn state. Note: it sometimes seems to be necessary to withdraw a window and then re-map it (e.g. with `wm deiconify`) to get some window managers to pay attention to changes in window attributes such as group.

---

**GEOMETRY MANAGEMENT**

By default a top-level window appears on the screen in its *natural size*, which is the one determined internally by its widgets and geometry managers. If the natural size of a top-level window changes, then the window's size changes to match. A top-level window can be given a size other than its natural size in two ways. First, the user can resize the window manually using the facilities of the window manager, such as resize handles. Second, the application can request a particular size
for a top-level window using the `wm geometry` command. These two cases are handled identically by Tk; in either case, the requested size overrides the natural size. You can return the window to its natural by invoking `wm geometry` with an empty `geometry` string.

Normally a top-level window can have any size from one pixel in each dimension up to the size of its screen. However, you can use the `wm minsize` and `wm maxsize` commands to limit the range of allowable sizes. The range set by `wm minsize` and `wm maxsize` applies to all forms of resizing, including the window's natural size as well as manual resizes and the `wm geometry` command. You can also use the command `wm resizable` to completely disable interactive resizing in one or both dimensions.

The `wm manage` and `wm forget` commands may be used to perform undocking and docking of windows. After a widget is managed by `wm manage` command, all other `wm` subcommands may be used with the widget. Only widgets created using the toplevel command may have an attached menu via the `-menu` configure option. A toplevel widget may be used as a frame and managed with any of the other geometry managers after using the `wm forget` command. Any menu associated with a toplevel widget will be hidden when managed by another geometry managers. The menus will reappear once the window is managed by `wm`. All custom bindtags for widgets in a subtree that have their top-level widget changed via a `wm manage` or `wm forget` command, must be redone to adjust any top-level widget path in the bindtags. Bindtags that have not been customized do not have to be redone.

**GRIDDED GEOMETRY MANAGEMENT**

Gridded geometry management occurs when one of the widgets of an application supports a range of useful sizes. This occurs, for example, in a text editor where the scrollbars, menus, and other adornments are fixed in size but the edit widget can support any number of lines of text or characters per line. In this case, it is usually desirable to let the user specify the number of lines or characters-per-line, either with the `wm geometry` command or by interactively resizing the window. In the case
of text, and in other interesting cases also, only discrete sizes of the window make sense, such as integral numbers of lines and characters-per-line; arbitrary pixel sizes are not useful.

Gridded geometry management provides support for this kind of application. Tk (and the window manager) assume that there is a grid of some sort within the application and that the application should be resized in terms of grid units rather than pixels. Gridded geometry management is typically invoked by turning on the setGrid option for a widget; it can also be invoked with the wm grid command or by calling Tk_SetGrid. In each of these approaches the particular widget (or sometimes code in the application as a whole) specifies the relationship between integral grid sizes for the window and pixel sizes. To return to non-gridded geometry management, invoke wm grid with empty argument strings.

When gridded geometry management is enabled then all the dimensions specified in wm minsize, wm maxsize, and wm geometry commands are treated as grid units rather than pixel units. Interactive resizing is also carried out in even numbers of grid units rather than pixels.

**BUGS**

Most existing window managers appear to have bugs that affect the operation of the wm command. For example, some changes will not take effect if the window is already active: the window will have to be withdrawn and de-iconified in order to make the change happen.

**EXAMPLES**

A fixed-size window that says that it is fixed-size too:

```bash
    toplevel .fixed
    wm title .fixed "Fixed-size Window"
    wm resizable .fixed 0 0
```
A simple dialog-like window, centred on the screen:

```python
# Create and arrange the dialog contents.
toplevel .msg
label .msg.l -text "This is a very simple dialog demo."
button .msg.ok -text OK -default active -command {destroy .msg}
pack .msg.ok -side bottom -fill x
pack .msg.l -expand 1 -fill both

# Now set the widget up as a centred dialog.

# But first, we need the geometry managers to finish up the interior of the dialog, for which we need to run the event loop with the widget hidden completely...
wm withdraw .msg
update
set x [expr {([winfo screenwidth .]-[winfo width .msg])/2}]
set y [expr {([winfo screenheight .]-[winfo height .msg])/2}]
wm geometry .msg +$x+$y
wm transient .msg
wm title .msg "Dialog demo"
wm deiconify .msg
```

SEE ALSO

toplevel, winfo

KEYWORDS

aspect ratio, deiconify, focus model, geometry, grid, group, icon, iconify, increments, position, size, title, top-level window, units, window manager

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1991-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
focus - Manage the input focus

SYNOPSIS

DESCRIPTION

The **focus** command is used to manage the Tk input focus. At any given time, one window on each display is designated as the **focus window**; any key press or key release events for the display are sent to that window. It is normally up to the window manager to redirect the focus among the top-level windows of a display. For example, some window managers automatically set the input focus to a top-level window whenever the mouse enters it; others redirect the input focus...
only when the user clicks on a window. Usually the window manager will set the focus only to top-level windows, leaving it up to the application to redirect the focus among the children of the top-level.

Tk remembers one focus window for each top-level (the most recent descendant of that top-level to receive the focus); when the window manager gives the focus to a top-level, Tk automatically redirects it to the remembered window. Within a top-level Tk uses an *explicit* focus model by default. Moving the mouse within a top-level does not normally change the focus; the focus changes only when a widget decides explicitly to claim the focus (e.g., because of a button click), or when the user types a key such as Tab that moves the focus.

The Tcl procedure `tk_focusFollowsMouse` may be invoked to create an *implicit* focus model: it reconfigures Tk so that the focus is set to a window whenever the mouse enters it. The Tcl procedures `tk_focusNext` and `tk_focusPrev` implement a focus order among the windows of a top-level; they are used in the default bindings for Tab and Shift-Tab, among other things.

The *focus* command can take any of the following forms:

**focus**

Returns the path name of the focus window on the display containing the application's main window, or an empty string if no window in this application has the focus on that display. Note: it is better to specify the display explicitly using `-displayof` (see below) so that the code will work in applications using multiple displays.

**focus window**

If the application currently has the input focus on window's display, this command resets the input focus for window's display to window and returns an empty string. If the application does not currently have the input focus on window's display, window will be remembered as the focus for its top-level; the next time the focus arrives at the top-level, Tk will redirect it to window. If window is an empty string then the command does nothing.
**focus -displayof** *window*

Returns the name of the focus window on the display containing *window*. If the focus window for *window*'s display is not in this application, the return value is an empty string.

**focus -force** *window*

Sets the focus of *window*'s display to *window*, even if the application does not currently have the input focus for the display. This command should be used sparingly, if at all. In normal usage, an application should not claim the focus for itself; instead, it should wait for the window manager to give it the focus. If *window* is an empty string then the command does nothing.

**focus -lastfor** *window*

Returns the name of the most recent window to have the input focus among all the windows in the same top-level as *window*. If no window in that top-level has ever had the input focus, or if the most recent focus window has been deleted, then the name of the top-level is returned. The return value is the window that will receive the input focus the next time the window manager gives the focus to the top-level.

### QUIRKS

When an internal window receives the input focus, Tk does not actually set the X focus to that window; as far as X is concerned, the focus will stay on the top-level window containing the window with the focus. However, Tk generates FocusIn and FocusOut events just as if the X focus were on the internal window. This approach gets around a number of problems that would occur if the X focus were actually moved; the fact that the X focus is on the top-level is invisible unless you use C code to query the X server directly.

### EXAMPLE

To make a window that only participates in the focus traversal ring when a variable is set, add the following bindings to the widgets *before* and *after* it in that focus ring:
button .before -text "Before"
button .middle -text "Middle"
button .after -text "After"
checkbutton .flag -variable traverseToMiddle -takefocus 0
pack .flag -side left
pack .before .middle .after
bind .before <Tab> {
    if {!$traverseToMiddle} {
        focus .after
        break
    }
}
bind .after <Shift-Tab> {
    if {!$traverseToMiddle} {
        focus .before
        break
    }
}
focus .before

KEYWORDS

events, focus, keyboard, top-level, window manager

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

option - Add/retrieve window options to/from the option database

SYNOPSIS

option add pattern value ?priority?
option clear
option get window name class
option readfile fileName ?priority?

DESCRIPTION

The option command allows you to add entries to the Tk option database or to retrieve options from the database. The add form of the command adds a new option to the database. Pattern contains the option being specified, and consists of names and/or classes separated by asterisks or dots, in the usual X format. Value contains a text string to associate with pattern; this is the value that will be returned in calls to Tk_GetOption or by invocations of the option get command. If priority is specified, it indicates the priority level for this option (see below for legal values); it defaults to interactive. This command always returns an empty string.

The option clear command clears the option database. Default options (from the RESOURCE_MANAGER property or the .Xdefaults file) will be reloaded automatically the next time an option is added to the database or removed from it. This command always returns an empty string.

The option get command returns the value of the option specified for window under name and class. If several entries in the option database
match *window*, *name*, and *class*, then the command returns whichever was created with highest *priority* level. If there are several matching entries at the same priority level, then it returns whichever entry was most recently entered into the option database. If there are no matching entries, then the empty string is returned.

The **readfile** form of the command reads *fileName*, which should have the standard format for an X resource database such as *.Xdefaults*, and adds all the options specified in that file to the option database. If *priority* is specified, it indicates the priority level at which to enter the options; *priority* defaults to **interactive**.

The *priority* arguments to the **option** command are normally specified symbolically using one of the following values:

**widgetDefault**
   Level 20. Used for default values hard-coded into widgets.

**startupFile**
   Level 40. Used for options specified in application-specific startup files.

**userDefault**
   Level 60. Used for options specified in user-specific defaults files, such as *.Xdefaults*, resource databases loaded into the X server, or user-specific startup files.

**interactive**
   Level 80. Used for options specified interactively after the application starts running. If *priority* is not specified, it defaults to this level.

Any of the above keywords may be abbreviated. In addition, priorities may be specified numerically using integers between 0 and 100, inclusive. The numeric form is probably a bad idea except for new priority levels other than the ones given above.

**EXAMPLES**
Instruct every button in the application to have red text on it unless explicitly overridden:

```python
option add *button.foreground red startupFile
```

Allow users to control what happens in an entry widget when the Return key is pressed by specifying a script in the option database and add a default option for that which rings the bell:

```python
entry .e
bind .e <Return> [option get .e returnCommand Command]
option add *.e.returnCommand bell widgetDefault
```

**KEYWORDS**

- database
- option
- priority
- retrieve

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
ttk::notebook - Multi-paned container widget

SYNOPSIS

DESCRIPTION

STANDARD OPTIONS
- class
- cursor, cursor, Cursor
- style
- takefocus, takeFocus, TakeFocus

WIDGET-SPECIFIC OPTIONS
- height, height, Height
- padding, padding, Padding
- width, width, Width

TAB OPTIONS
- state, state, State
- sticky, sticky, Sticky
- padding, padding, Padding
- text, text, Text
- image, image, Image
- compound, compound, Compound
- underline, underline, Underline

TAB IDENTIFIERS

WIDGET COMMAND

pathname add window ?options...?
pathname configure ?options?
pathname cget option
pathname forget tabid
pathname hide tabid
pathname identify x y
pathname index tabid
pathname insert pos subwindow options...
**NAME**

ttk::notebook - Multi-paned container widget

**SYNOPSIS**

```bash
ttk::notebook pathName ?options...?
pathName add window ?options...?
pathName insert index window ?options...?
```

**DESCRIPTION**

A `ttk::notebook` widget manages a collection of windows and displays a single one at a time. Each slave window is associated with a `tab`, which the user may select to change the currently-displayed window.

**STANDARD OPTIONS**

- `-class`
- `-cursor, cursor, Cursor`
- `-style`
- `-takefocus, takeFocus, TakeFocus`

**WIDGET-SPECIFIC OPTIONS**

Command-Line Name: `-height`
Database Name: `height`
Database Class: **Height**
If present and greater than zero, specifies the desired height of the pane area (not including internal padding or tabs). Otherwise, the maximum height of all panes is used.

Command-Line Name: `-padding`
Database Name: **padding**
Database Class: **Padding**
Specifies the amount of extra space to add around the outside of the notebook. The padding is a list of up to four length specifications `left top right bottom`. If fewer than four elements are specified, `bottom` defaults to `top`, `right` defaults to `left`, and `top` defaults to `left`.

Command-Line Name: `-width`
Database Name: **width**
Database Class: **Width**
If present and greater than zero, specifies the desired width of the pane area (not including internal padding). Otherwise, the maximum width of all panes is used.

**TAB OPTIONS**

The following options may be specified for individual notebook panes:

Command-Line Name: `-state`
Database Name: **state**
Database Class: **State**
Either `normal`, `disabled` or `hidden`. If `disabled`, then the tab is not selectable. If `hidden`, then the tab is not shown.

Command-Line Name: `-sticky`
Database Name: **sticky**
Database Class: **Sticky**
Specifies how the slave window is positioned within the pane area. Value is a string containing zero or more of the characters `n, s, e,` or `w`. Each letter refers to a side (north, south, east, or west) that the slave window will “stick” to, as per the `grid` geometry manager.
Command-Line Name: **-padding**
Database Name: **padding**
Database Class: **Padding**
  Specifies the amount of extra space to add between the notebook and this pane. Syntax is the same as for the widget **-padding** option.

Command-Line Name: **-text**
Database Name: **text**
Database Class: **Text**
  Specifies a string to be displayed in the tab.

Command-Line Name: **-image**
Database Name: **image**
Database Class: **Image**
  Specifies an image to display in the tab. See *ttk_widget(n)* for details.

Command-Line Name: **-compound**
Database Name: **compound**
Database Class: **Compound**
  Specifies how to display the image relative to the text, in the case both **-text** and **-image** are present. See *label(n)* for legal values.

Command-Line Name: **-underline**
Database Name: **underline**
Database Class: **Underline**
  Specifies the integer index (0-based) of a character to underline in the text string. The underlined character is used for mnemonic activation if *ttk::notebook::enableTraversal* is called.

---

**TAB IDENTIFIERS**

The *tabid* argument to the following commands may take any of the following forms:

- An integer between zero and the number of tabs;
• The name of a slave window;

• A positional specification of the form “@x,y”, which identifies the tab

• The literal string “current”, which identifies the currently-selected tab; or:

• The literal string “end”, which returns the number of tabs (only valid for “pathname index”).

### WIDGET COMMAND

**pathname** *add* *window* ?*options*...?

Adds a new tab to the notebook. See **TAB OPTIONS** for the list of available *options*. If *window* is currently managed by the notebook but hidden, it is restored to its previous position.

**pathname** *configure* ?*options*?

See **ttk::widget(n)**.

**pathname** *cget* *option*

See **ttk::widget(n)**.

**pathname** *forget* *tabid*

Removes the tab specified by *tabid*, unmaps and unmanages the associated window.

**pathname** *hide* *tabid*

Hides the tab specified by *tabid*. The tab will not be displayed, but the associated window remains managed by the notebook and its configuration remembered. Hidden tabs may be restored with the *add* command.

**pathname** *identify* *x* *y*

Returns the name of the element at position *x*, *y*. See **ttk::widget(n)**.

**pathname** *index* *tabid*

Returns the numeric index of the tab specified by *tabid*, or the total
number of tabs if \textit{tabid} is the string \textit{end}.

\textit{pathname insert pos subwindow options}...
Inserts a pane at the specified position. \textit{pos} is either the string \textit{end}, an integer index, or the name of a managed subwindow. If \textit{subwindow} is already managed by the notebook, moves it to the specified position. See \textbf{TAB OPTIONS} for the list of available options.

\textit{pathname instate statespec ?script}...
See \texttt{ttk::widget(n)}.

\textit{pathname select ?tabid}?
Selects the specified tab. The associated slave window will be displayed, and the previously-selected window (if different) is unmapped. If \textit{tabid} is omitted, returns the widget name of the currently selected pane.

\textit{pathname state ?statespec}?
See \texttt{ttk::widget(n)}.

\textit{pathname tab tabid |-option ?value ...}
Query or modify the options of the specific tab. If no \textit{-option} is specified, returns a dictionary of the tab option values. If one \textit{-option} is specified, returns the value of that \textit{option}. Otherwise, sets the \textit{-options} to the corresponding \textit{values}. See \textbf{TAB OPTIONS} for the available options.

\textit{pathname tabs}
Returns the list of windows managed by the notebook.

\textbf{KEYBOARD TRAVERSAL}

To enable keyboard traversal for a toplevel window containing a notebook widget \$nb, call:

\begin{verbatim}
  ttk::notebook::enableTraversal $nb
\end{verbatim}
This will extend the bindings for the toplevel window containing the notebook as follows:

- **Control-Tab** selects the tab following the currently selected one.
- **Shift-Control-Tab** selects the tab preceding the currently selected one.
- **Alt-K**, where K is the mnemonic (underlined) character of any tab, will select that tab.

Multiple notebooks in a single toplevel may be enabled for traversal, including nested notebooks. However, notebook traversal only works properly if all panes are direct children of the notebook.

**VIRTUAL EVENTS**

The notebook widget generates a <<NotebookTabChanged>> virtual event after a new tab is selected.

**EXAMPLE**

```
pack [ttk::notebook .nb]
    .nb add [frame .nb.f1] -text "First tab"
    .nb add [frame .nb.f2] -text "Second tab"
    .nb select .nb.f2
ttk::notebook::enableTraversal .nb
```

**SEE ALSO**

ttk::widget, grid

**KEYWORDS**

pane, tab

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
Tcl_Alloc, Tcl_Free, Tcl_Realloc, Tcl_AttemptAlloc,
Tcl_AttemptRealloc, ckalloc, ckfree, ckrealloc, attemptckalloc,
attemptckrealloc - allocate or free heap memory

SYNOPSIS
#include <tcl.h>
char *
Tcl_Alloc(size)
void
Tcl_Free(ptr)
char *
Tcl_Realloc(ptr, size)
char *
Tcl_AttemptAlloc(size)
char *
Tcl_AttemptRealloc(ptr, size)
char *
ckalloc(size)
void
ckfree(ptr)
char *
ckrealloc(ptr, size)
char *
attemptckalloc(size)
char *
attemptckrealloc(ptr, size)

ARGUMENTS
DESCRIPTION
KEYWORDS
Tcl_Alloc, Tcl_Free, Tcl_Realloc, Tcl_AttemptAlloc, Tcl_AttemptRealloc, ckalloc, ckfree, ckrealloc, attemptckalloc, attemptckrealloc - allocate or free heap memory

SYNOPSIS

#include <tcl.h>

char * Tcl_Alloc(size)
void Tcl_Free(ptr)
char * Tcl_Realloc(ptr, size)
char * Tcl_AttemptAlloc(size)
char * Tcl_AttemptRealloc(ptr, size)
char * ckalloc(size)
void ckfree(ptr)
char * ckrealloc(ptr, size)
char * attemptckalloc(size)
char * attemptckrealloc(ptr, size)

ARGUMENTS

unsigned int size (in)  Size in bytes of the memory block to allocate.
char *ptr (in)          Pointer to memory block to free or realloc.

DESCRIPTION
These procedures provide a platform and compiler independent interface for memory allocation. Programs that need to transfer ownership of memory blocks between Tcl and other modules should use these routines rather than the native `malloc()` and `free()` routines provided by the C run-time library.

**Tcl_Alloc** returns a pointer to a block of at least size bytes suitably aligned for any use.

**Tcl_Free** makes the space referred to by `ptr` available for further allocation.

**Tcl_Realloc** changes the size of the block pointed to by `ptr` to `size` bytes and returns a pointer to the new block. The contents will be unchanged up to the lesser of the new and old sizes. The returned location may be different from `ptr`. If `ptr` is NULL, this is equivalent to calling **Tcl_Alloc** with just the `size` argument.

**Tcl_AttemptAlloc** and **Tcl_AttemptRealloc** are identical in function to **Tcl_Alloc** and **Tcl_Realloc**, except that **Tcl_AttemptAlloc** and **Tcl_AttemptRealloc** will not cause the Tcl interpreter to panic if the memory allocation fails. If the allocation fails, these functions will return NULL. Note that on some platforms, but not all, attempting to allocate a zero-sized block of memory will also cause these functions to return NULL.

The procedures **ckalloc**, **ckfree**, **ckrealloc**, **attemptckalloc**, and **attemptckrealloc** are implemented as macros. Normally, they are synonyms for the corresponding procedures documented on this page. When Tcl and all modules calling Tcl are compiled with **TCL_MEM_DEBUG** defined, however, these macros are redefined to be special debugging versions of these procedures. To support Tcl's memory debugging within a module, use the macros rather than direct calls to **Tcl_Alloc**, etc.

**KEYWORDS**
alloc, allocation, free, malloc, memory, realloc, TCL_MEM_DEBUG
NAME
Tcl_FSRegister, Tcl_FSUnregister, Tcl_FSDATA,
Tcl_FSMountsChanged, Tcl_FSGetFileSystemForPath,
Tcl_FSGetAddressType, Tcl_FSCopyFile, Tcl_FSCopyDirectory,
Tcl_FSCreateDirectory, Tcl_FSDeleteFile,
Tcl_FSRemoveDirectory, Tcl_FSRenameFile,
Tcl_FSListsVolumes, Tcl_FSEvalFile, Tcl_FSEvalFileEx,
Tcl_FSLoadFile, Tcl_FSMatchInDirectory, Tcl_FSLink,
Tcl_FSLstat, Tcl_FSUtime, Tcl_FSFileAttrsGet,
Tcl_FSFileAttrsSet, Tcl_FSFileAttrStrings, Tcl_FSStat,
Tcl_FSAccess, Tcl_FSOpenFileChannel, Tcl_FSGetCwd,
Tcl_FSShdir, Tcl_FSPathSeparator, Tcl_FSJoinPath,
Tcl_FSSplitPath, Tcl_FSEqualPaths,
Tcl_FSGetNormalizedPath, Tcl_FSJoinToPath,
Tcl_FSConvertToPathType, Tcl_FSGetInternalRep,
Tcl_FSGetTranslatedPath, Tcl_FSGetTranslatedStringPath,
Tcl_FSNativePath, Tcl_FSGetNativePath,
Tcl_FSFsSystemInfo, Tcl_AllocStatBuf - procedures to interact with any filesystem
SYNOPSIS
#include <tcl.h>
int
Tcl_FSRegister(clientData, fsPtr)
int
Tcl_FSUnregister(fsPtr)
ClientData
Tcl_FSDATA(fsPtr)
void
Tcl_FSMountsChanged(fsPtr)
Tcl_FsFileSystemForPath(pathPtr)
Tcl_PathType
Tcl_FSGetPathType(pathPtr)
int
Tcl_FSCopyFile(srcPathPtr, destPathPtr)
int
Tcl_FSCopyDirectory(srcPathPtr, destPathPtr, errorPtr)
int
Tcl_FSCreateDirectory(pathPtr)
int
Tcl_FSDeleteFile(pathPtr)
int
Tcl_FSRemoveDirectory(pathPtr, int recursive, errorPtr)
int
Tcl_FSRenameFile(srcPathPtr, destPathPtr)
Tcl_Obj*
Tcl_FSLstatVolumes(void)
int
Tcl_FSEvalFileEx(interp, pathPtr, encodingName)
int
Tcl_FSEvalFile(interp, pathPtr)
int
Tcl_FSLoadFile(interp, pathPtr, sym1, sym2, proc1Ptr,
proc2Ptr,
handlePtr, unloadProcPtr)
int
Tcl_FSMatchInDirectory(interp, resultPtr, pathPtr, pattern,
types)
Tcl_Obj*
Tcl_FSLstat(pathPtr, statPtr)
int
Tcl_FSUtime(pathPtr, tval)
int
Tcl_FSFileAttrsGet(interp, int index, pathPtr, objPtrRef)
int
Tcl_FSFileAttrsSet(interp, int index, pathPtr, Tcl_Obj *objPtr)
const char**
Tcl_FSFileAttrStrings(pathPtr, objPtrRef)
int
Tcl_FSStat(pathPtr, statPtr)
int
Tcl_FSAccess(pathPtr, mode)
Tcl_Channel
Tcl_FSOpenFileChannel(interp, pathPtr, modeString, permissions)
Tcl_Obj*
Tcl_FSGetCwd(interp)
int
Tcl_FSChdir(pathPtr)
Tcl_Obj*
Tcl_FSPathSeparator(pathPtr)
Tcl_Obj*
Tcl_FSJoinPath(listObj, elements)
Tcl_Obj*
Tcl_FSSplitPath(pathPtr, lenPtr)
int
Tcl_FSEqualPaths(firstPtr, secondPtr)
Tcl_Obj*
Tcl_FSGetNormalizedPath(interp, pathPtr)
Tcl_Obj*
Tcl_FSJoinToPath(basePtr, objc, objv)
int
Tcl_FSConvertToPathType(interp, pathPtr)
ClientData
Tcl_FSGetInternalRep(pathPtr, fsPtr)
Tcl_Obj *
Tcl_FSGetTranslatedPath(interp, pathPtr)
const char *
Tcl_FSGetTranslatedStringPath(interp, pathPtr)
Tcl_Obj*
Tcl_FSNewNativePath(fsPtr, clientData)
const char *
Tcl_FSGetNativePath(pathPtr)
Tcl_Obj*
Tcl_FSFileSystemInfo(pathPtr)
Tcl_StatBuf*
Tcl_AllocStatBuf()

ARGUMENTS
DESCRIPTION
FS API FUNCTIONS
THE VIRTUAL FILESYSTEM API
THE TCL_FILESYSTEM STRUCTURE
EXAMPLE FILESYSTEM DEFINITION
FILESYSTEM INFRASTRUCTURE
TYPENAME
STRUCTURE LENGTH
VERSION
PATHINFILESYSTEMPROC
DUPINTERNALREPPROC
FREEINTERNALREPPROC
INTERNALTONORMALIZEDPROC
CREATEINTERNALREPPROC
NORMALIZEPATHPROC
FILESYSTEM OPERATIONS
FILESYSTEMPATHTYPEPROC
FILESYSTEMSEPARATORPROC
STATPROC
ACCESSPROC
OPENFILECHANNELPROC
MATCHINDIRECTORYPROC
UTIMEPROC
LINKPROC
LISTVOLUMESPROC
FILEATTRSTRINGSPROC
FILEATTRSGETPROC
FILEATTRSSETPROC
CREATEDIRECTORYPROC
REMOVEDIRECTORYPROC
DELETEFILEPROC
FILESYSTEM EFFICIENCY
NAME

Tcl_FSRegister, Tcl_FSUnregister, Tcl_FSData, Tcl_FSMountsChanged, Tcl_FSGetFileSystemForPath, Tcl_FSGetPathType, Tcl_FSCopyFile, Tcl_FSCopyDirectory, Tcl_FSCreateDirectory, Tcl_FSDeleteFile, Tcl_FSRemoveDirectory, Tcl_FSRenameFile, Tcl_FSListVolumes, Tcl_FSEvalFile, Tcl_FSEvalFileEx, Tcl_FSLoadFile, Tcl_FSMatchInDirectory, Tcl_FSLink, Tcl_FSLstat, Tcl_FSUtime, Tcl_FSFileAttrsGet, Tcl_FSFileAttrsSet, Tcl_FSFileAttrStrings, Tcl_FSSStat, Tcl_FSAccess, Tcl_FSOpenFileChannel, Tcl_FSGetCwd, Tcl_FSChdir, Tcl_FSPathSeparator, Tcl_FSLjoinPath, Tcl_FSSplitPath, Tcl_FSEqualPaths, Tcl_FSGetNormalizedPath, Tcl_FSLjoinToPath, Tcl_FSCConvertToPathType, Tcl_FSGetInternalRep, Tcl_FSGetTranslatedPath, Tcl_FSGetTranslatedStringPath, Tcl_FSNewNativePath, Tcl_FSGetNativePath, Tcl_FSFileSystemInfo, Tcl_AllocStatBuf - procedures to interact with any filesystem

SYNOPSIS

#include <tcl.h>
int
Tcl_FSRegister(clientData, fsPtr)
int
Tcl_FSUnregister(fsPtr)
ClientData
Tcl_FSData(fsPtr)
void
Tcl_FSMountsChanged(fsPtr)
Tcl_Filesystem*
Tcl_FSGetFileSystemForPath(pathPtr)
Tcl_PathType
Tcl_FSGetPathType(pathPtr)
int
Tcl_FSCopyFile(srcPathPtr, destPathPtr)
int
Tcl_FSCopyDirectory(srcPathPtr, destPathPtr, errorPtr)
int
Tcl_FSCreateDirectory(pathPtr)
int
Tcl_FSDeleteFile(pathPtr)
int
Tcl_FSRemoveDirectory(pathPtr, int recursive, errorPtr)
int
Tcl_FSRenameFile(srcPathPtr, destPathPtr)
Tcl_Obj*
Tcl_FSListVolumes(void)
int
Tcl_FSEvalFileEx(interp, pathPtr, encodingName)
int
Tcl_FSEvalFile(interp, pathPtr)
int
Tcl_FSLoadFile(interp, pathPtr, sym1, sym2, proc1Ptr, proc2Ptr, handlePtr, unloadProcPtr)
int
Tcl_FSMatchInDirectory(interp, resultPtr, pathPtr, pattern, types)
Tcl_Obj*
Tcl_FSLink(linkNamePtr, toPtr, linkAction)
int
Tcl_FSLstat(pathPtr, statPtr)
int
Tcl_FSUtime(pathPtr, tv)
int
Tcl_FSFileAttrssGet(interp, int index, pathPtr, objPtrRef)
int
Tcl_FSFileAttrsSet(interp, int index, pathPtr, Tcl_Obj *objPtr)
const char**
Tcl_FSFileAttrStrings(pathPtr, objPtrRef)
int
Tcl_FSStat(pathPtr, statPtr)
int
Tcl_FSAccess(pathPtr, mode)
Tcl_Channel
Tcl_FSOpenFileChannel(interp, pathPtr, modeString, permissions)
Tcl_Obj*
Tcl_FSGetCwd(interp)
int
Tcl_FSChdir(pathPtr)
Tcl_Obj*
Tcl_FSPathSeparator(pathPtr)
Tcl_Obj*
Tcl_FSJoinPath(listObj, elements)
Tcl_Obj*
Tcl_FSSplitPath(pathPtr, lenPtr)
int
Tcl_FSEqualPaths(firstPtr, secondPtr)
Tcl_Obj*
Tcl_FSGetNormalizedPath(interp, pathPtr)
Tcl_Obj*
Tcl_FSJoinToPath(basePtr, objc, objv)
int
Tcl_FSCreateToPathType(interp, pathPtr)
ClientData
Tcl_FSGetInternalRep(pathPtr, fsPtr)
Tcl_Obj *
Tcl_FSGetTranslatedPath(interp, pathPtr)
const char *
Tcl_FSGetTranslatedStringPath(interp, pathPtr)
Tcl_Obj*
Tcl_FSNewNativePath(fsPtr, clientData)
const char *
Tcl_FSGetNativePath(pathPtr)
### TCL Files

**Tcl_Obj**

**Tcl_FSFileSystemInfo**(*pathPtr*)

**Tcl_StatBuf**

**Tcl_AllocStatBuf**()

### ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_Filesystem <em>fsPtr (in)</em></em></td>
<td>Points to a structure containing the addresses of procedures that can be called to perform the various filesystem operations.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>pathPtr (in)</em></em></td>
<td>The path represented by this object is used for the operation in question. If the object does not already have an internal path representation, it will be converted to have one.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>srcPathPtr (in)</em></em></td>
<td>As for <em>pathPtr</em>, but used for the source file for a copy or rename operation.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>destPathPtr (in)</em></em></td>
<td>As for <em>pathPtr</em>, but used for the destination filename for a copy or rename operation.</td>
</tr>
<tr>
<td>*<em>const char <em>encodingName (in)</em></em></td>
<td>The encoding of the data stored in the file identified by <em>pathPtr</em> and to be evaluated.</td>
</tr>
<tr>
<td>*<em>const char <em>pattern (in)</em></em></td>
<td>Only files or directories</td>
</tr>
</tbody>
</table>
Only files or directories matching the type descriptions contained in this structure will be returned. This parameter may be NULL.

Interpreter to use either for results, evaluation, or reporting error messages.

The native description of the path object to create.

The first of two path objects to compare. The object may be converted to path type.

The second of two path objects to compare. The object may be converted to path type.

The list of path elements to operate on with a join operation.

If non-negative, the number of elements in the listObj which should be joined together. If negative, then all elements are joined.
<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>errorPtr</strong> (out)</td>
<td>In the case of an error, filled with an object containing the name of the file which caused an error in the various copy/rename operations.</td>
</tr>
<tr>
<td><strong>objPtrRef</strong> (out)</td>
<td>Filled with an object containing the result of the operation.</td>
</tr>
<tr>
<td><strong>resultPtr</strong> (out)</td>
<td>Pre-allocated object in which to store (using <code>Tcl_ListObjAppendElement</code>) the list of files or directories which are successfully matched.</td>
</tr>
<tr>
<td>int <code>mode</code> (in)</td>
<td>Mask consisting of one or more of <code>R_OK</code>, <code>W_OK</code>, <code>X_OK</code> and <code>F_OK</code>. <code>R_OK</code>, <code>W_OK</code> and <code>X_OK</code> request checking whether the file exists and has read, write and execute permissions, respectively. <code>F_OK</code> just requests checking for the existence of the file.</td>
</tr>
<tr>
<td><code>statPtr</code> (out)</td>
<td>The structure that contains the result of a <code>stat</code> or <code>lstat</code> operation.</td>
</tr>
<tr>
<td>const char *<code>sym1</code> (in)</td>
<td>Name of a procedure to look up in the file's symbol table</td>
</tr>
<tr>
<td>Variable Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>const char *sym2 (in)</td>
<td>Name of a procedure to look up in the file's symbol table</td>
</tr>
<tr>
<td>Tcl_PackageInitProc **proc1Ptr (out)</td>
<td>Filled with the init function for this code.</td>
</tr>
<tr>
<td>Tcl_PackageInitProc **proc2Ptr (out)</td>
<td>Filled with the safe-init function for this code.</td>
</tr>
<tr>
<td>ClientData *clientDataPtr (out)</td>
<td>Filled with the clientData value to pass to this code's unload function when it is called.</td>
</tr>
<tr>
<td>Tcl_LoadHandle *handlePtr (out)</td>
<td>Filled with an abstract token representing the loaded file.</td>
</tr>
<tr>
<td>Tcl_FSUnloadFileProc **unloadProcPtr (out)</td>
<td>Filled with the function to use to unload this piece of code.</td>
</tr>
<tr>
<td>utimbuf *tval (in)</td>
<td>The access and modification times in this structure are read and used to set those values for a given file.</td>
</tr>
<tr>
<td>const char *modeString (in)</td>
<td>Specifies how the file is to be accessed. May have any of the values allowed for the mode argument to the Tcl open command.</td>
</tr>
<tr>
<td>int permissions (in)</td>
<td>POSIX-style permission flags such as 0644. If a</td>
</tr>
</tbody>
</table>
new file is created, these permissions will be set on the created file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *lenPtr (out)</td>
<td>If non-NULL, filled with the number of elements in the split path.</td>
</tr>
<tr>
<td>Tcl_Obj *basePtr (in)</td>
<td>The base path on to which to join the given elements. May be NULL.</td>
</tr>
<tr>
<td>int objc (in)</td>
<td>The number of elements in objv.</td>
</tr>
<tr>
<td>Tcl_Obj *const objv[] (in)</td>
<td>The elements to join to the given base path.</td>
</tr>
<tr>
<td>Tcl_Obj *linkNamePtr (in)</td>
<td>The name of the link to be created or read.</td>
</tr>
<tr>
<td>Tcl_Obj *toPtr (in)</td>
<td>What the link called linkNamePtr should be linked to, or NULL if the symbolic link specified by linkNamePtr is to be read.</td>
</tr>
<tr>
<td>int linkAction (in)</td>
<td>OR-ed combination of flags indicating what kind of link should be created (will be ignored if toPtr is NULL). Valid bits to set are TCL_CREATE_SYMBOLIC and TCL_CREATE_HARD_LINK. When both flags are set and the underlying...</td>
</tr>
</tbody>
</table>
There are several reasons for calling the **Tcl_FS** API functions (e.g. **Tcl_FSAccess** and **Tcl_FSStat**) rather than calling system level functions like **access** and **stat** directly. First, they will work cross-platform, so an extension which calls them should work unmodified on Unix and Windows. Second, the Windows implementation of some of these functions fixes some bugs in the system level calls. Third, these function calls deal with any “Utf to platform-native” path conversions which may be required (and may cache the results of such conversions for greater efficiency on subsequent calls). Fourth, and perhaps most importantly, all of these functions are “virtual filesystem aware”. Any virtual filesystem (VFS for short) which has been registered (through **Tcl_FSRegister**) may reroute file access to alternative media or access methods. This means that all of these functions (and therefore the corresponding **file**, **glob**, **pwd**, **cd**, **open**, etc. Tcl commands) may be operate on “files” which are not native files in the native filesystem. This also means that any Tcl extension which accesses the filesystem (FS for short) through this API is automatically “virtual filesystem aware”. Of course, if an extension accesses the native filesystem directly (through platform-specific APIs, for example), then Tcl cannot intercept such calls.

If appropriate VFSes have been registered, the “files” may, to give two examples, be remote (e.g. situated on a remote ftp server) or archived (e.g. lying inside a .zip archive). Such registered filesystems provide a lookup table of functions to implement all or some of the functionality listed here. Finally, the **Tcl_FSStat** and **Tcl_FSLstat** calls abstract away from what the “struct stat” buffer is actually declared to be, allowing the same code to be used both on systems with and systems without support for files larger than 2GB in size.

The **Tcl_FS** API is objectified and may cache internal representations
and other path-related strings (e.g. the current working directory). One side-effect of this is that one must not pass in objects with a reference count of zero to any of these functions. If such calls were handled, they might result in memory leaks (under some circumstances, the filesystem code may wish to retain a reference to the passed in object, and so one must not assume that after any of these calls return, the object still has a reference count of zero - it may have been incremented) or in a direct segmentation fault (or other memory access error) due to the object being freed part way through the complex object manipulation required to ensure that the path is fully normalized and absolute for filesystem determination. The practical lesson to learn from this is that

```c
Tcl_Obj *path = Tcl_NewStringObj(...);
Tcl_FSWhatever(path);
Tcl_DecrRefCount(path);
```

is wrong, and may cause memory errors. The path must have its reference count incremented before passing it in, or decrementing it. For this reason, objects with a reference count of zero are considered not to be valid filesystem paths and calling any Tcl_FS API function with such an object will result in no action being taken.

## FS API FUNCTIONS

**Tcl_FSCopyFile** attempts to copy the file given by `srcPathPtr` to the path name given by `destPathPtr`. If the two paths given lie in the same filesystem (according to `Tcl_FSGetFileSystemForPath`) then that filesystem's “copy file” function is called (if it is non-NULL). Otherwise the function returns -1 and sets the `errno` global C variable to the “EXDEV” POSIX error code (which signifies a “cross-domain link”).

**Tcl_FSCopyDirectory** attempts to copy the directory given by `srcPathPtr` to the path name given by `destPathPtr`. If the two paths given lie in the same filesystem (according to `Tcl_FSGetFileSystemForPath`) then that filesystem's “copy file”
function is called (if it is non-NULL). Otherwise the function returns -1 and sets the `errno` global C variable to the “EXDEV” POSIX error code (which signifies a “cross-domain link”).

**Tcl_FSCreateDirectory** attempts to create the directory given by `pathPtr` by calling the owning filesystem's “create directory” function.

**Tcl_FSDeleteFile** attempts to delete the file given by `pathPtr` by calling the owning filesystem's “delete file” function.

**Tcl_FSRenameFile** attempts to rename the file or directory given by `srcPathPtr` to the path name given by `destPathPtr`. If the two paths given lie in the same filesystem (according to `Tcl_FSGetFileSystemForPath`) then that filesystem's “rename file” function is called (if it is non-NULL). Otherwise the function returns -1 and sets the `errno` global C variable to the “EXDEV” POSIX error code (which signifies a “cross-domain link”).

**Tcl_FSListVolumes** calls each filesystem which has a non-NULL “list volumes” function and asks them to return their list of root volumes. It accumulates the return values in a list which is returned to the caller (with a reference count of 0).

**Tcl_FSEvalFileEx** reads the file given by `pathPtr` using the encoding identified by `encodingName` and evaluates its contents as a Tcl script. It returns the same information as `Tcl_EvalObjEx`. If `encodingName` is NULL, the system encoding is used for reading the file contents. If the file could not be read then a Tcl error is returned to describe why the file could not be read. The eofchar for files is “\32” (^Z) for all platforms. If you require a “\^Z” in code for string comparison, you can use “\032” or “\u001a”, which will be safely substituted by the Tcl interpreter into “\^Z”. **Tcl_FSEvalFile** is a simpler version of `Tcl_FSEvalFileEx` that always uses the system encoding when reading the file.

**Tcl_FSLoadFile** dynamically loads a binary code file into memory and
returns the addresses of two procedures within that file, if they are defined. The appropriate function for the filesystem to which *pathPtr* belongs will be called. If that filesystem does not implement this function (most virtual filesystems will not, because of OS limitations in dynamically loading binary code), Tcl will attempt to copy the file to a temporary directory and load that temporary file.

Returns a standard Tcl completion code. If an error occurs, an error message is left in the interp's result.

**Tcl_FSMatchInDirectory** is used by the globbing code to search a directory for all files which match a given pattern. The appropriate function for the filesystem to which *pathPtr* belongs will be called.

The return value is a standard Tcl result indicating whether an error occurred in globbing. Error messages are placed in interp (unless interp is NULL, which is allowed), but good results are placed in the resultPtr given.

Note that the **glob** code implements recursive patterns internally, so this function will only ever be passed simple patterns, which can be matched using the logic of **string match**. To handle recursion, Tcl will call this function frequently asking only for directories to be returned. A special case of being called with a NULL pattern indicates that the path needs to be checked only for the correct type.

**Tcl_FSLink** replaces the library version of **readlink**, and extends it to support the creation of links. The appropriate function for the filesystem to which *linkNamePtr* belongs will be called.

If the *toPtr* is NULL, a “read link” action is performed. The result is a Tcl_Obj specifying the contents of the symbolic link given by *linkNamePtr*, or NULL if the link could not be read. The result is owned by the caller, which should call Tcl_DecrRefCount when the result is no longer needed. If the *toPtr* is not NULL, Tcl should create a link of one of the types passed in in the **linkAction** flag. This flag is an ORed combination of **TCL_CREATE_SYMBOLIC_LINK** and **TCL_CREATE_HARD_LINK**. Where a choice exists (i.e. more than one
flag is passed in), the Tcl convention is to prefer symbolic links. When a link is successfully created, the return value should be toPtr (which is therefore already owned by the caller). If unsuccessful, NULL is returned.

Tcl_FSLstat fills the stat structure statPtr with information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file. The stat structure includes info regarding device, inode (always 0 on Windows), privilege mode, nlink (always 1 on Windows), user id (always 0 on Windows), group id (always 0 on Windows), rdev (same as device on Windows), size, last access time, last modification time, and creation time.

If path exists, Tcl_FSLstat returns 0 and the stat structure is filled with data. Otherwise, -1 is returned, and no stat info is given.

Tcl_FSUtime replaces the library version of utime.

This returns 0 on success and -1 on error (as per the utime documentation). If successful, the function will update the “atime” and “mtime” values of the file given.

Tcl_FSFileAttrsGet implements read access for the hookable file attributes subcommand. The appropriate function for the filesystem to which pathPtr belongs will be called.

If the result is TCL_OK, then an object was placed in objPtrRef, which will only be temporarily valid (unless Tcl_IncrRefCount is called).

Tcl_FSFileAttrsSet implements write access for the hookable file attributes subcommand. The appropriate function for the filesystem to which pathPtr belongs will be called.

Tcl_FSFileAttrStrings implements part of the hookable file attributes subcommand. The appropriate function for the filesystem to which pathPtr belongs will be called.

The called procedure may either return an array of strings, or may
instead return NULL and place a Tcl list into the given objPtrRef. Tcl will take that list and first increment its reference count before using it. On completion of that use, Tcl will decrement its reference count. Hence if the list should be disposed of by Tcl when done, it should have a reference count of zero, and if the list should not be disposed of, the filesystem should ensure it retains a reference count to the object.

**Tcl_FSAccess** checks whether the process would be allowed to read, write or test for existence of the file (or other filesystem object) whose name is *pathname*. If *pathname* is a symbolic link on Unix, then permissions of the file referred by this symbolic link are tested.

On success (all requested permissions granted), zero is returned. On error (at least one bit in mode asked for a permission that is denied, or some other error occurred), -1 is returned.

**Tcl_FSStat** fills the stat structure statPtr with information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file. The stat structure includes info regarding device, inode (always 0 on Windows), privilege mode, nlink (always 1 on Windows), user id (always 0 on Windows), group id (always 0 on Windows), rdev (same as device on Windows), size, last access time, last modification time, and creation time.

If *path* exists, **Tcl_FSStat** returns 0 and the stat structure is filled with data. Otherwise, -1 is returned, and no stat info is given.

**Tcl_FSOpenFileChannel** opens a file specified by *pathPtr* and returns a channel handle that can be used to perform input and output on the file. This API is modeled after the fopen procedure of the Unix standard I/O library. The syntax and meaning of all arguments is similar to those given in the Tcl **open** command when opening a file. If an error occurs while opening the channel, **Tcl_FSOpenFileChannel** returns NULL and records a POSIX error code that can be retrieved with **Tcl_GetErrno**. In addition, if *interp* is non-NULL, **Tcl_FSOpenFileChannel** leaves an error message in *interp*'s result after any error.
The newly created channel is not registered in the supplied interpreter; to register it, use `Tcl_RegisterChannel`, described below. If one of the standard channels, `stdin`, `stdout` or `stderr` was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

**Tcl_FSGetCwd** replaces the library version of `getcwd`.

It returns the Tcl library's current working directory. This may be different to the native platform's working directory, which happens when the current working directory is not in the native filesystem.

The result is a pointer to a Tcl_Obj specifying the current directory, or NULL if the current directory could not be determined. If NULL is returned, an error message is left in the `interp`'s result.

The result already has its reference count incremented for the caller. When it is no longer needed, that reference count should be decremented. This is needed for thread-safety purposes, to allow multiple threads to access this and related functions, while ensuring the results are always valid.

**Tcl_FSCChdir** replaces the library version of `chdir`. The path is normalized and then passed to the filesystem which claims it. If that filesystem does not implement this function, Tcl will fallback to a combination of `stat` and `access` to check whether the directory exists and has appropriate permissions.

For results, see `chdir` documentation. If successful, we keep a record of the successful path in `cwdPathPtr` for subsequent calls to `Tcl_FSGetCwd`.

**Tcl_FSPathSeparator** returns the separator character to be used for most specific element of the path specified by `pathPtr` (i.e. the last part of the path).

The separator is returned as a Tcl_Obj containing a string of length 1. If the path is invalid, NULL is returned.
**Tcl_FSJoinPath** takes the given Tcl_Obj, which must be a valid list (which is allowed to have a reference count of zero), and returns the path object given by considering the first *elements* elements as valid path segments (each path segment may be a complete path, a partial path or just a single possible directory or file name). If any path segment is actually an absolute path, then all prior path segments are discarded. If *elements* is less than 0, we use the entire list.

It is possible that the returned object is actually an element of the given list, so the caller should be careful to increment the reference count of the result before freeing the list.

The returned object, typically with a reference count of zero (but it could be shared under some conditions), contains the joined path. The caller must add a reference count to the object before using it. In particular, the returned object could be an element of the given list, so freeing the list might free the object prematurely if no reference count has been taken. If the number of elements is zero, then the returned object will be an empty-string Tcl_Obj.

**Tcl_FSSplitPath** takes the given Tcl_Obj, which should be a valid path, and returns a Tcl list object containing each segment of that path as an element. It returns a list object with a reference count of zero. If the passed in *lenPtr* is non-NULL, the variable it points to will be updated to contain the number of elements in the returned list.

**Tcl_FSEqualPaths** tests whether the two paths given represent the same filesystem object

It returns 1 if the paths are equal, and 0 if they are different. If either path is NULL, 0 is always returned.

**Tcl_FSGetNormalizedPath** this important function attempts to extract from the given Tcl_Obj a unique normalized path representation, whose string value can be used as a unique identifier for the file.

It returns the normalized path object, owned by Tcl, or NULL if the path was invalid or could otherwise not be successfully converted. Extraction
of absolute, normalized paths is very efficient (because the filesystem operates on these representations internally), although the result when the filesystem contains numerous symbolic links may not be the most user-friendly version of a path. The return value is owned by Tcl and has a lifetime equivalent to that of the pathPtr passed in (unless that is a relative path, in which case the normalized path object may be freed any time the cwd changes) - the caller can of course increment the refCount if it wishes to maintain a copy for longer.

**Tcl_FSJoinToPath** takes the given object, which should usually be a valid path or NULL, and joins onto it the array of paths segments given.

Returns object, typically with refCount of zero (but it could be shared under some conditions), containing the joined path. The caller must add a refCount to the object before using it. If any of the objects passed into this function (pathPtr or path elements) have a refCount of zero, they will be freed when this function returns.

**Tcl_FSConvertToPathType** tries to convert the given Tcl_Obj to a valid Tcl path type, taking account of the fact that the cwd may have changed even if this object is already supposedly of the correct type. The filename may begin with “~” (to indicate current user's home directory) or “~<user>” (to indicate any user's home directory).

If the conversion succeeds (i.e. the object is a valid path in one of the current filesystems), then **TCL_OK** is returned. Otherwise **TCL_ERROR** is returned, and an error message may be left in the interpreter.

**Tcl_FSGetInternalRep** extracts the internal representation of a given path object, in the given filesystem. If the path object belongs to a different filesystem, we return NULL. If the internal representation is currently NULL, we attempt to generate it, by calling the filesystem's **Tcl_FSCreateInternalRepProc**.

Returns NULL or a valid internal path representation. This internal representation is cached, so that repeated calls to this function will not require additional conversions.
Tcl_FSGetTranslatedPath attempts to extract the translated path from the given Tcl_Obj.

If the translation succeeds (i.e. the object is a valid path), then it is returned. Otherwise NULL will be returned, and an error message may be left in the interpreter. A “translated” path is one which contains no “~” or “~user” sequences (these have been expanded to their current representation in the filesystem). The object returned is owned by the caller, which must store it or call Tcl_DecrRefCount to ensure memory is freed. This function is of little practical use, and Tcl_FSGetNormalizedPath or Tcl_GetNativePath are usually better functions to use for most purposes.

Tcl_FSGetTranslatedStringPath does the same as Tcl_FSGetTranslatedPath, but returns a character string or NULL. The string returned is dynamically allocated and owned by the caller, which must store it or call ckfree to ensure it is freed. Again, Tcl_FSGetNormalizedPath or Tcl_GetNativePath are usually better functions to use for most purposes.

Tcl_FSNewNativePath performs something like the reverse of the usual obj->path->nativerep conversions. If some code retrieves a path in native form (from, e.g. readlink or a native dialog), and that path is to be used at the Tcl level, then calling this function is an efficient way of creating the appropriate path object type.

The resulting object is a pure “path” object, which will only receive a UTF-8 string representation if that is required by some Tcl code.

Tcl_FSGetNativePath is for use by the Win/Unix native filesystems, so that they can easily retrieve the native (char* or TCHAR*) representation of a path. This function is a convenience wrapper around Tcl_FSGetInternalRep, and assumes the native representation is string-based. It may be desirable in the future to have non-string-based native representations (for example, on MacOSX, a representation using a fileSpec of FSRef structure would probably be more efficient). On Windows a full Unicode representation would allow for paths of unlimited length. Currently the representation is simply a character
string which may contain either the relative path or a complete, absolute normalized path in the native encoding (complex conditions dictate which of these will be provided, so neither can be relied upon, unless the path is known to be absolute). If you need a native path which must be absolute, then you should ask for the native version of a normalized path. If for some reason a non-absolute, non-normalized version of the path is needed, that must be constructed separately (e.g. using Tcl_FSGetTranslatedPath).

The native representation is cached so that repeated calls to this function will not require additional conversions. The return value is owned by Tcl and has a lifetime equivalent to that of the pathPtr passed in (unless that is a relative path, in which case the native representation may be freed any time the cwd changes).

Tcl_FSFileSystemInfo returns a list of two elements. The first element is the name of the filesystem (e.g. “native”, “vfs”, “zip”, or “prowrap”, perhaps), and the second is the particular type of the given path within that filesystem (which is filesystem dependent). The second element may be empty if the filesystem does not provide a further categorization of files.

A valid list object is returned, unless the path object is not recognized, when NULL will be returned.

Tcl_FSGetFileSystemForPath returns the a pointer to the Tcl_Filesystem which accepts this path as valid.

If no filesystem will accept the path, NULL is returned.

Tcl_FSGetPathType determines whether the given path is relative to the current directory, relative to the current volume, or absolute.

It returns one of TCL_PATH_ABSOLUTE, TCL_PATH_RELATIVE, or TCL_PATH_VOLUME_RELATIVE

Tcl_AllocStatBuf allocates a Tcl_StatBuf on the system heap (which may be deallocated by being passed to ckfree.) This allows extensions
to invoke **Tcl_FStat** and **Tcl_FSLStat** without being dependent on the size of the buffer. That in turn depends on the flags used to build Tcl.

### THE VIRTUAL FILESYSTEM API

A filesystem provides a **Tcl_Filesystem** structure that contains pointers to functions that implement the various operations on a filesystem; these operations are invoked as needed by the generic layer, which generally occurs through the functions listed above.

The **Tcl_Filesystem** structures are manipulated using the following methods.

**Tcl_FSRegister** takes a pointer to a filesystem structure and an optional piece of data to associated with that filesystem. On calling this function, Tcl will attach the filesystem to the list of known filesystems, and it will become fully functional immediately. Tcl does not check if the same filesystem is registered multiple times (and in general that is not a good thing to do). **TCL_OK** will be returned.

**Tcl_FSUnregister** removes the given filesystem structure from the list of known filesystems, if it is known, and returns **TCL_OK**. If the filesystem is not currently registered, **TCL_ERROR** is returned.

**Tcl_FSData** will return the ClientData associated with the given filesystem, if that filesystem is registered. Otherwise it will return NULL.

**Tcl_FSMountsChanged** is used to inform the Tcl's core that the set of mount points for the given (already registered) filesystem have changed, and that cached file representations may therefore no longer be correct.

### THE TCL_FILESYSTEM STRUCTURE

The **Tcl_Filesystem** structure contains the following fields:

```c
typedef struct Tcl_Filesystem {
```
const char *typeName;
int structureLength;
Tcl_FSVersion version;
Tcl_FSPathInFilesystemProc *pathInFilesystemProc
Tcl_FSDupInternalRepProc *dupInternalRepProc;
Tcl_FSFreeInternalRepProc *freeInternalRepProc;
Tcl_FSInternalToNormalizedProc *internalToNormalizedProc
Tcl_FSCreateInternalRepProc *createInternalRepProc
Tcl_FSNormalizePathProc *normalizePathProc;
Tcl_FSFilesystemPathTypeProc *filesystemPathTypeProc
Tcl_FSFilesystemSeparatorProc *filesystemSeparatorProc
Tcl_FSStatProc *statProc;
Tcl_FSAccessProc *accessProc;
Tcl_FSOpenFileChannelProc *openFileChannelProc;
Tcl_FSMatchInDirectoryProc *matchInDirectoryProc
Tcl_FSUtimeProc *utimeProc;
Tcl_FSLLinkProc *linkProc;
Tcl_FSLListVolumesProc *listVolumesProc;
Tcl_FSFileAttrStringsProc *fileAttrStringsProc;
Tcl_FSFileAttrsGetProc *fileAttrsGetProc;
Tcl_FSFileAttrsSetProc *fileAttrsSetProc;
Tcl_FSCreateDirectoryProc *createDirectoryProc;
Tcl_FSRemoveDirectoryProc *removeDirectoryProc;
Tcl_FSDeleleFileProc *deleteFileProc;
Tcl_FSCopyFileProc *copyFileProc;
Tcl_FSRenameFileProc *renameFileProc;
Tcl_FSCopyDirectoryProc *copyDirectoryProc;
Tcl_FSLstatProc *lstatProc;
Tcl_FSLoadFileProc *loadFileProc;
Tcl_FSGetCwdProc *getCwdProc;
Tcl_FSChdirProc *chdirProc;
} Tcl_Filesystem;

Except for the first three fields in this structure which contain simple data elements, all entries contain addresses of functions called by the generic filesystem layer to perform the complete range of filesystem
related actions.

The many functions in this structure are broken down into three categories: infrastructure functions (almost all of which must be implemented), operational functions (which must be implemented if a complete filesystem is provided), and efficiency functions (which need only be implemented if they can be done so efficiently, or if they have side-effects which are required by the filesystem; Tcl has less efficient emulations it can fall back on). It is important to note that, in the current version of Tcl, most of these fallbacks are only used to handle commands initiated in Tcl, not in C. What this means is, that if a `file rename` command is issued in Tcl, and the relevant filesystem(s) do not implement their `Tcl_FSRenameFileProc`, Tcl's core will instead fallback on a combination of other filesystem functions (it will use `Tcl_FSCopyFileProc` followed by `Tcl_FSDeleteFileProc`, and if `Tcl_FSCopyFileProc` is not implemented there is a further fallback). However, if a `Tcl_FSRenameFileProc` command is issued at the C level, no such fallbacks occur. This is true except for the last four entries in the filesystem table (`lstat`, `load`, `getcwd` and `chdir`) for which fallbacks do in fact occur at the C level.

Any functions which take path names in `Tcl_Obj` form take those names in UTF-8 form. The filesystem infrastructure API is designed to support efficient, cached conversion of these UTF-8 paths to other native representations.

EXAMPLE FILESYSTEM DEFINITION

Here is the filesystem lookup table used by the “vfs” extension which allows filesystem actions to be implemented in Tcl.

```c
static Tcl_Filesystem vfsFilesystem = {
    "tclvfs",
    sizeof(Tcl_Filesystem),
    TCL_FILESYSTEM_VERSION_1,
    &VfsPathInFilesystem,
    &VfsDupInternalRep,
```
&VfsFreeInternalRep,
/* No internal to normalized, since we don't create
 any pure 'internal' Tcl_Obj path representations */
NULL,
/* No create native rep function, since we don't
 it and don't choose to support uses of
 * Tcl_FSNewNativePath */
NULL,
/* Normalize path isn't needed - we assume paths
 have one representation */
NULL,
&VfsFilesystemPathType,
&VfsFilesystemSeparator,
&VfsStat,
&VfsAccess,
&VfsOpenFileChannel,
&VfsMatchInDirectory,
&VfsUtime,
/* We choose not to support symbolic links inside
 * VFS's */
NULL,
&VfsListVolumes,
&VfsFileAttrStrings,
&VfsFileAttrsGet,
&VfsFileAttrsSet,
&VfsCreateDirectory,
&VfsRemoveDirectory,
&VfsDeleteFile,
/* No copy file; use the core fallback mechanism */
NULL,
/* No rename file; use the core fallback mechanism */
NULL,
/* No copy directory; use the core fallback mechanism */
NULL,
/* Core will use stat for lstat */
NULL,
/* No load; use the core fallback mechanism */
FILESYSTEM INFRASTRUCTURE

These fields contain basic information about the filesystem structure and addresses of functions which are used to associate a particular filesystem with a file path, and deal with the internal handling of path representations, for example copying and freeing such representations.

**TYPENAME**

The `typeName` field contains a null-terminated string that identifies the type of the filesystem implemented, e.g. “native”, “zip” or “vfs”.

**STRUCTURE LENGTH**

The `structureLength` field is generally implemented as `sizeof(Tcl_Filesystem)`, and is there to allow easier binary backwards compatibility if the size of the structure changes in a future Tcl release.

**VERSION**

The `version` field should be set to `TCL_FILESYSTEM_VERSION_1`.

**PATHINFOFILESYSTEMPROC**

The `pathInfoFilesystemProc` field contains the address of a function which is called to determine whether a given path object belongs to this filesystem or not. Tcl will only call the rest of the filesystem functions with a path for which this function has returned `TCL_OK`. If the path does not belong, -1 should be returned (the behaviour of Tcl for any
other return value is not defined). If **TCL_OK** is returned, then the optional *clientDataPtr* output parameter can be used to return an internal (filesystem specific) representation of the path, which will be cached inside the path object, and may be retrieved efficiently by other filesystem functions. Tcl will simultaneously cache the fact that this path belongs to this filesystem. Such caches are invalidated when filesystem structures are added or removed from Tcl's internal list of known filesystems.

```c
typedef int Tcl_FSPathInFilesystemProc(
    Tcl_Obj *pathPtr,
    ClientData *clientDataPtr);
```

**DUPINTERNALREPPROC**

This function makes a copy of a path's internal representation, and is called when Tcl needs to duplicate a path object. If NULL, Tcl will simply not copy the internal representation, which may then need to be regenerated later.

```c
typedef ClientData Tcl_FSDupInternalRepProc(
    ClientData clientData);
```

**FREEINTERNALREPPROC**

Free the internal representation. This must be implemented if internal representations need freeing (i.e. if some memory is allocated when an internal representation is generated), but may otherwise be NULL.

```c
typedef void Tcl_FSFreeInternalRepProc(
    ClientData clientData);
```
INTERNALTONORMALIZEDPROC

Function to convert internal representation to a normalized path. Only required if the filesystem creates pure path objects with no string/path representation. The return value is a Tcl object whose string representation is the normalized path.

```c
typedef Tcl_Obj* Tcl_FSInternalToNormalizedProc(
    ClientData clientData);
```

CREATEINTERNALREPPROC

Function to take a path object, and calculate an internal representation for it, and store that native representation in the object. May be NULL if paths have no internal representation, or if the `Tcl_FSPathInFilesystemProc` for this filesystem always immediately creates an internal representation for paths it accepts.

```c
typedef ClientData Tcl_FSCreateInternalRepProc(
    Tcl_Obj *pathPtr);
```

NORMALIZEPATHPROC

Function to normalize a path. Should be implemented for all filesystems which can have multiple string representations for the same path object. In Tcl, every “path” must have a single unique “normalized” string representation. Depending on the filesystem, there may be more than one unnormalized string representation which refers to that path (e.g. a relative path, a path with different character case if the filesystem is case insensitive, a path contain a reference to a home directory such as “~”, a path containing symbolic links, etc). If the very last component in the path is a symbolic link, it should not be converted into the object it points to (but its case or other aspects should be made unique). All other path components should be converted from symbolic links. This
one exception is required to agree with Tcl's semantics with **file delete**, **file rename**, **file copy** operating on symbolic links. This function may be called with **nextCheckpoint** either at the beginning of the path (i.e. zero), at the end of the path, or at any intermediate file separator in the path. It will never point to any other arbitrary position in the path. In the last of the three valid cases, the implementation can assume that the path up to and including the file separator is known and normalized.

```c
typedef int Tcl_FSNormalizePathProc(
    Tcl_Interp *interp,
    Tcl_Obj *pathPtr,
    int nextCheckpoint);
```

**FILESYSTEM OPERATIONS**

The fields in this section of the structure contain addresses of functions which are called to carry out the basic filesystem operations. A filesystem which expects to be used with the complete standard Tcl command set must implement all of these. If some of them are not implemented, then certain Tcl commands may fail when operating on paths within that filesystem. However, in some instances this may be desirable (for example, a read-only filesystem should not implement the last four functions, and a filesystem which does not support symbolic links need not implement the **readlink** function, etc. The Tcl core expects filesystems to behave in this way).

**FILESYSTEMPATHTYPEPROC**

Function to determine the type of a path in this filesystem. May be NULL, in which case no type information will be available to users of the filesystem. The “type” is used only for informational purposes, and should be returned as the string representation of the Tcl_Obj which is returned. A typical return value might be “networked”, “zip” or “ftp”. The Tcl_Obj result is owned by the filesystem and so Tcl will increment the refCount of that object if it wishes to retain a reference to it.
typedef Tcl_Obj* Tcl_FSFilesystemPathTypeProc(
    Tcl_Obj *pathPtr);

FILESYSTEMSEPARATORPROC

Function to return the separator character(s) for this filesystem. This need only be implemented if the filesystem wishes to use a different separator than the standard string “/”. Amongst other uses, it is returned by the file separator command. The return value should be an object with refCount of zero.

typedef Tcl_Obj* Tcl_FSFilesystemSeparatorProc(
    Tcl_Obj *pathPtr);

STATPROC

Function to process a Tcl_FSStat call. Must be implemented for any reasonable filesystem, since many Tcl level commands depend crucially upon it (e.g. file atime, file isdirectory, file size, glob).

defined int Tcl_FSStatProc(
    Tcl_Obj *pathPtr,
    Tcl_StatBuf *statPtr);

The Tcl_FSStatProc fills the stat structure statPtr with information about the specified file. You do not need any access rights to the file to get this information but you need search rights to all directories named in the path leading to the file. The stat structure includes info regarding device, inode (always 0 on Windows), privilege mode, nlink (always 1 on Windows), user id (always 0 on Windows), group id (always 0 on Windows), rdev (same as device on Windows), size, last access time, last modification time, and creation time.
If the file represented by `pathPtr` exists, the `Tcl_FSStatProc` returns 0 and the stat structure is filled with data. Otherwise, -1 is returned, and no stat info is given.

### ACCESSPROC

Function to process a `Tcl_FSAccess` call. Must be implemented for any reasonable filesystem, since many Tcl level commands depend crucially upon it (e.g. `file exists`, `file readable`).

```c
typedef int Tcl_FSAccessProc(
    Tcl_Obj *pathPtr,
    int mode);
```

The `Tcl_FSAccessProc` checks whether the process would be allowed to read, write or test for existence of the file (or other filesystem object) whose name is in `pathPtr`. If the pathname refers to a symbolic link, then the permissions of the file referred by this symbolic link should be tested.

On success (all requested permissions granted), zero is returned. On error (at least one bit in mode asked for a permission that is denied, or some other error occurred), -1 is returned.

### OPENFILECHANNELPROC

Function to process a `Tcl_FSOpenFileChannel` call. Must be implemented for any reasonable filesystem, since any operations which require open or accessing a file's contents will use it (e.g. `open`, `encoding`, and many Tk commands).

```c
typedef Tcl_Channel Tcl_FSOpenFileChannelProc(
    Tcl_Interp *interp,
    Tcl_Obj *pathPtr,
    int mode,
```
The **Tcl_FSOpenFileChannelProc** opens a file specified by `pathPtr` and returns a channel handle that can be used to perform input and output on the file. This API is modeled after the `fopen` procedure of the Unix standard I/O library. The syntax and meaning of all arguments is similar to those given in the Tcl `open` command when opening a file, where the `mode` argument is a combination of the POSIX flags `O_RDONLY`, `O_WRONLY`, etc. If an error occurs while opening the channel, the **Tcl_FSOpenFileChannelProc** returns NULL and records a POSIX error code that can be retrieved with `Tcl_GetErrno`. In addition, if `interp` is non-NULL, the **Tcl_FSOpenFileChannelProc** leaves an error message in `interp`'s result after any error.

The newly created channel is not registered in the supplied interpreter; to register it, use **Tcl_RegisterChannel**. If one of the standard channels, `stdin`, `stdout` or `stderr` was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

### MATCHINDIRECTORYPROC

Function to process a **Tcl_FSMatchInDirectory** call. If not implemented, then glob and recursive copy functionality will be lacking in the filesystem (and this may impact commands like `encoding names` which use glob functionality internally).

```c
typedef int Tcl_FSMatchInDirectoryProc(
    Tcl_Interp* interp,
    Tcl_Obj* resultPtr,
    Tcl_Obj* pathPtr,
    const char* pattern,
    Tcl_GlobTypeData* types);
```

The function should return all files or directories (or other filesystem
objects) which match the given pattern and accord with the types specification given. There are two ways in which this function may be called. If pattern is NULL, then pathPtr is a full path specification of a single file or directory which should be checked for existence and correct type. Otherwise, pathPtr is a directory, the contents of which the function should search for files or directories which have the correct type. In either case, pathPtr can be assumed to be both non-NULL and non-empty. It is not currently documented whether pathPtr will have a file separator at its end of not, so code should be flexible to both possibilities.

The return value is a standard Tcl result indicating whether an error occurred in the matching process. Error messages are placed in interp, unless interp in NULL in which case no error message need be generated; on a TCL_OK result, results should be added to the resultPtr object given (which can be assumed to be a valid unshared Tcl list). The matches added to resultPtr should include any path prefix given in pathPtr (this usually means they will be absolute path specifications). Note that if no matches are found, that simply leads to an empty result; errors are only signaled for actual file or filesystem problems which may occur during the matching process.

The Tcl_GlobTypeData structure passed in the types parameter contains the following fields:

```c
typedef struct Tcl_GlobTypeData {
    /* Corresponds to bcdpfls as in 'find -t' */
    int type;
    /* Corresponds to file permissions */
    int perm;
    /* Acceptable mac type */
    Tcl_Obj *macType;
    /* Acceptable mac creator */
    Tcl_Obj *macCreator;
} Tcl_GlobTypeData;
```
There are two specific cases which it is important to handle correctly, both when types is non-NULL. The two cases are when types->types & TCL_GLOB_TYPE_DIR or types->types & TCL_GLOB_TYPE_MOUNT are true (and in particular when the other flags are false). In the first of these cases, the function must list the contained directories. Tcl uses this to implement recursive globbing, so it is critical that filesystems implement directory matching correctly. In the second of these cases, with TCL_GLOB_TYPE_MOUNT, the filesystem must list the mount points which lie within the given pathPtr (and in this case, pathPtr need not lie within the same filesystem - different to all other cases in which this function is called). Support for this is critical if Tcl is to have seamless transitions between from one filesystem to another.

**UTIMEPROC**

Function to process a Tcl_FSUtime call. Required to allow setting (not reading) of times with file mtime, file atime and the open-r/open-w/fcopy implementation of file copy.

```c
typedef int Tcl_FSUtimeProc(
    Tcl_Obj *pathPtr,
    struct utimbuf *tval);
```

The access and modification times of the file specified by pathPtr should be changed to the values given in the tval structure.

The return value should be 0 on success and -1 on an error, as with the system utime.

**LINKPROC**

Function to process a Tcl_FSLink call. Should be implemented only if the filesystem supports links, and may otherwise be NULL.

```c
typedef Tcl_Obj* Tcl_FSLinkProc(
```
Tcl_Obj *linkNamePtr,
Tcl_Obj *toPtr,
int linkAction);

If toPtr is NULL, the function is being asked to read the contents of a link. The result is a Tcl_Obj specifying the contents of the link given by linkNamePtr, or NULL if the link could not be read. The result is owned by the caller (and should therefore have its ref count incremented before being returned). Any callers should call Tcl_DecrRefCount on this result when it is no longer needed. If toPtr is not NULL, the function should attempt to create a link. The result in this case should be toPtr if the link was successful and NULL otherwise. In this case the result is not owned by the caller (i.e. no ref count manipulation on either end is needed). See the documentation for Tcl_FSLink for the correct interpretation of the linkAction flags.

**LISTVOLUMESPROC**

Function to list any filesystem volumes added by this filesystem. Should be implemented only if the filesystem adds volumes at the head of the filesystem, so that they can be returned by file volumes.

typedef Tcl_Obj* Tcl_FSListVolumesProc(void);

The result should be a list of volumes added by this filesystem, or NULL (or an empty list) if no volumes are provided. The result object is considered to be owned by the filesystem (not by Tcl's core), but should be given a refCount for Tcl. Tcl will use the contents of the list and then decrement that refCount. This allows filesystems to choose whether they actually want to retain a “master list” of volumes or not (if not, they generate the list on the fly and pass it to Tcl with a refCount of 1 and then forget about the list, if yes, then they simply increment the refCount of their master list and pass it to Tcl which will copy the contents and then decrement the count back to where it was).
Therefore, Tcl considers return values from this proc to be read-only.

**FILEATTRSTRINGSPROC**

Function to list all attribute strings which are valid for this filesystem. If not implemented the filesystem will not support the file attributes command. This allows arbitrary additional information to be attached to files in the filesystem. If it is not implemented, there is no need to implement the get and set methods.

```c
typedef const char** Tcl_FSFileAttrStringsProc(
    Tcl_Obj *pathPtr,
    Tcl_Obj** objPtrRef);
```

The called function may either return an array of strings, or may instead return NULL and place a Tcl list into the given objPtrRef. Tcl will take that list and first increment its reference count before using it. On completion of that use, Tcl will decrement its reference count. Hence if the list should be disposed of by Tcl when done, it should have a reference count of zero, and if the list should not be disposed of, the filesystem should ensure it returns an object with a reference count of at least one.

**FILEATTRSGETPROC**

Function to process a Tcl_FSFileAttrsGet call, used by file attributes.

```c
typedef int Tcl_FSFileAttrsGetProc(
    Tcl_Interp *interp,
    int index,
    Tcl_Obj *pathPtr,
    Tcl_Obj **objPtrRef);
```

Returns a standard Tcl return code. The attribute value retrieved, which
corresponds to the index'th element in the list returned by the \texttt{Tcl\_FSFileAttrStringsProc}, is a \texttt{Tcl\_Obj} placed in \texttt{objPtrRef} (if \texttt{TCL\_OK} was returned) and is likely to have a reference count of zero. Either way we must either store it somewhere (e.g. the Tcl result), or Incr/Decr its reference count to ensure it is properly freed.

**FILEATTRSSETPROC**

Function to process a \texttt{Tcl\_FSFileAttrSet} call, used by file attributes. If the filesystem is read-only, there is no need to implement this.

\begin{verbatim}
typedef int Tcl\_FSFileAttrSetProc(
    Tcl\_Interp *interp,
    int index,
    Tcl\_Obj *pathPtr,
    Tcl\_Obj *objPtr);
\end{verbatim}

The attribute value of the index'th element in the list returned by the \texttt{Tcl\_FSFileAttrStringsProc} should be set to the \texttt{objPtr} given.

**CREATEDIRECTORYPROC**

Function to process a \texttt{Tcl\_FSCreateDirectory} call. Should be implemented unless the FS is read-only.

\begin{verbatim}
typedef int Tcl\_FSCreateDirectoryProc(
    Tcl\_Obj *pathPtr);
\end{verbatim}

The return value is a standard Tcl result indicating whether an error occurred in the process. If successful, a new directory should have been added to the filesystem in the location specified by \texttt{pathPtr}.

**REMOVEDIRECTORYPROC**
Function to process a **Tcl_FSRemoveDirectory** call. Should be implemented unless the FS is read-only.

```c
typedef int Tcl_FSRemoveDirectoryProc(
    Tcl_Obj *pathPtr,
    int recursive,
    Tcl_Obj **errorPtr);
```

The return value is a standard Tcl result indicating whether an error occurred in the process. If successful, the directory specified by `pathPtr` should have been removed from the filesystem. If the `recursive` flag is given, then a non-empty directory should be deleted without error. If this flag is not given, then and the directory is non-empty a POSIX “EEXIST” error should be signalled. If an error does occur, the name of the file or directory which caused the error should be placed in `errorPtr`.

### DELETEFILEPROC

Function to process a **Tcl_FSDeleteFile** call. Should be implemented unless the FS is read-only.

```c
typedef int Tcl_FSDeleteFileProc(
    Tcl_Obj *pathPtr);
```

The return value is a standard Tcl result indicating whether an error occurred in the process. If successful, the file specified by `pathPtr` should have been removed from the filesystem. Note that, if the filesystem supports symbolic links, Tcl will always call this function and not `Tcl_FSRemoveDirectoryProc` when needed to delete them (even if they are symbolic links to directories).

### FILESYSTEM EFFICIENCY

These functions need not be implemented for a particular filesystem
because the core has a fallback implementation available. See each individual description for the consequences of leaving the field NULL.

**LSTATPROC**

Function to process a **Tcl_FSLstat** call. If not implemented, Tcl will attempt to use the **statProc** defined above instead. Therefore it need only be implemented if a filesystem can differentiate between **stat** and **lstat** calls.

```c
typedef int Tcl_FSLstatProc(
    Tcl_Obj *pathPtr,
    Tcl_StatBuf *statPtr);
```

The behavior of this function is very similar to that of the **Tcl_FSSStatProc** defined above, except that if it is applied to a symbolic link, it returns information about the link, not about the target file.

**COPYFILEPROC**

Function to process a **Tcl_FSCopyFile** call. If not implemented Tcl will fall back on **open**-r, **open**-w and **fcopy** as a copying mechanism. Therefore it need only be implemented if the filesystem can perform that action more efficiently.

```c
typedef int Tcl_FSCopyFileProc(
    Tcl_Obj *srcPathPtr,
    Tcl_Obj *destPathPtr);
```

The return value is a standard Tcl result indicating whether an error occurred in the copying process. Note that, **destPathPtr** is the name of the file which should become the copy of **srcPathPtr**. It is never the name of a directory into which **srcPathPtr** could be copied (i.e. the function is much simpler than the Tcl level **file copy** subcommand).
Note that, if the filesystem supports symbolic links, Tcl will always call this function and not `copyDirectoryProc` when needed to copy them (even if they are symbolic links to directories). Finally, if the filesystem determines it cannot support the `file copy` action, calling `Tcl_SetErrno(EXDEV)` and returning a non-TCL_OK result will tell Tcl to use its standard fallback mechanisms.

## RENAMEFILEPROC

Function to process a `Tcl_FSRenameFile` call. If not implemented, Tcl will fall back on a copy and delete mechanism. Therefore it need only be implemented if the filesystem can perform that action more efficiently.

```c
typedef int Tcl_FSRenameFileProc(
    Tcl_Obj *srcPathPtr,
    Tcl_Obj *destPathPtr);
```

The return value is a standard Tcl result indicating whether an error occurred in the renaming process. If the filesystem determines it cannot support the `file rename` action, calling `Tcl_SetErrno(EXDEV)` and returning a non-TCL_OK result will tell Tcl to use its standard fallback mechanisms.

## COPYDIRECTORYPROC

Function to process a `Tcl_FSCopyDirectory` call. If not implemented, Tcl will fall back on a recursive `file mkdir, file copy` mechanism. Therefore it need only be implemented if the filesystem can perform that action more efficiently.

```c
typedef int Tcl_FSCopyDirectoryProc(
    Tcl_Obj *srcPathPtr,
    Tcl_Obj *destPathPtr,
    Tcl_Obj **errorPtr);
```
The return value is a standard Tcl result indicating whether an error occurred in the copying process. If an error does occur, the name of the file or directory which caused the error should be placed in errorPtr. Note that, destPathPtr is the name of the directory-name which should become the mirror-image of srcPathPtr. It is not the name of a directory into which srcPathPtr should be copied (i.e. the function is much simpler than the Tcl level file copy subcommand). Finally, if the filesystem determines it cannot support the directory copy action, calling Tcl_SetErrno(EXDEV) and returning a non-TCL_OK result will tell Tcl to use its standard fallback mechanisms.

**LOADFILEPROC**

Function to process a Tcl_FSLoadFile call. If not implemented, Tcl will fall back on a copy to native-temp followed by a Tcl_FSLoadFile on that temporary copy. Therefore it need only be implemented if the filesystem can load code directly, or it can be implemented simply to return TCL_ERROR to disable load functionality in this filesystem entirely.

```c
typedef int Tcl_FSLoadFileProc(
    Tcl_Interp *interp,
    Tcl_Obj *pathPtr,
    Tcl_LoadHandle *handlePtr,
    Tcl_FSUnloadFileProc *unloadProcPtr);
```

Returns a standard Tcl completion code. If an error occurs, an error message is left in the interp's result. The function dynamically loads a binary code file into memory. On a successful load, the handlePtr should be filled with a token for the dynamically loaded file, and the unloadProcPtr should be filled in with the address of a procedure. The unload procedure will be called with the given Tcl_LoadHandle as its only parameter when Tcl needs to unload the file. For example, for the native filesystem, the Tcl_LoadHandle returned is currently a token
which can be used in the private **TclpFindSymbol** to access functions in the new code. Each filesystem is free to define the **Tcl_LoadHandle** as it requires. Finally, if the filesystem determines it cannot support the file load action, calling **Tcl_SetErrno(EXDEV)** and returning a non-**TCL_OK** result will tell Tcl to use its standard fallback mechanisms.

---

**UNLOADFILEPROC**

Function to unload a previously successfully loaded file. If load was implemented, then this should also be implemented, if there is any cleanup action required.

```c
typedef void Tcl_FSUnloadFileProc(
    Tcl_LoadHandle loadHandle);
```

**GETCWDPROC**

Function to process a **Tcl_FSGetCwd** call. Most filesystems need not implement this. It will usually only be called once, if **getcwd** is called before **chdir**. May be NULL.

```c
typedef Tcl_Obj* Tcl_FSGetCwdProc(
    Tcl_Interp *interp);
```

If the filesystem supports a native notion of a current working directory (which might perhaps change independent of Tcl), this function should return that cwd as the result, or NULL if the current directory could not be determined (e.g. the user does not have appropriate permissions on the cwd directory). If NULL is returned, an error message is left in the **interp**'s result.

**CHDIRPROC**

Function to process a **Tcl_FSChdir** call. If filesystems do not implement
this, it will be emulated by a series of directory access checks. Otherwise, virtual filesystems which do implement it need only respond with a positive return result if the pathPtr is a valid, accessible directory in their filesystem. They need not remember the result, since that will be automatically remembered for use by Tcl_FSGetCwd. Real filesystems should carry out the correct action (i.e. call the correct system chdir API).

```c
typedef int Tcl_FSChdirProc(
  Tcl_Obj *pathPtr);
```

The Tcl_FSChdirProc changes the applications current working directory to the value specified in pathPtr. The function returns -1 on error or 0 on success.

**SEE ALSO**

cd, file, load, open, pwd, unload

**KEYWORDS**

stat, access, filesystem, vfs, virtual
NAME
Tcl_UniChar, Tcl_UniCharCaseMatch, Tcl_UniCharNcasecmp, Tcl_UniCharToUtf, Tcl_UtfToUniChar,
Tcl_UniCharToUtfDString, Tcl_UtfToUniCharDString,
Tcl_UniCharLen, Tcl_UniCharNcmp, Tcl_UtfCharComplete,
Tcl_NumUtfChars, Tcl_UtfFindFirst, Tcl_UtfFindLast,
Tcl_UtfNext, Tcl_UtfPrev, Tcl_UniCharAtIndex, Tcl_UtfAtIndex,
Tcl_UtfBackslash - routines for manipulating UTF-8 strings

SYNOPSIS
#include <tcl.h>
typedef ... Tcl_UniChar;
int
Tcl_UniCharToUtf(ch, buf)
int
Tcl_UtfToUniChar(src, chPtr)
char *
Tcl_UniCharToUtfDString(uniStr, uniLength, dsPtr)
Tcl_UniChar *
Tcl_UtfToUniCharDString(src, length, dsPtr)
int
Tcl_UniCharLen(uniStr)
int
Tcl_UniCharNcmp(ucs, uct, numChars)
int
Tcl_UniCharNcasecmp(ucs, uct, numChars)
int
Tcl_UniCharCaseMatch(uniStr, uniPattern, nocase)
int
Tcl_UtfNcmp(cs, ct, numChars)
int
Tcl_UtfNcasecmp(cs, ct, numChars)
int
Tcl_UtfCharComplete(src, length)
int
Tcl_NumUtfChars(src, length)
const char *
Tcl_UtfFindFirst(src, ch)
const char *
Tcl_UtfFindLast(src, ch)
const char *
Tcl_UtfNext(src)
const char *
Tcl_UtfPrev(src, start)
Tcl_UniChar
Tcl_UniCharAtIndex(src, index)
const char *
Tcl_UtfAtIndex(src, index)
int
Tcl_UtfBackslash(src, readPtr, dst)

**ARGUMENTS**

**DESCRIPTION**

**KEYWORDS**

<table>
<thead>
<tr>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_UniChar, Tcl_UniCharCaseMatch, Tcl_UniCharNcasecmp, Tcl_UniCharToUtf, Tcl_UtfToUniChar, Tcl_UniCharToUtfDString, Tcl_UtfToUniCharDString, Tcl_UniCharLen, Tcl_UniCharNcmp, Tcl_UtfCharComplete, Tcl_NumUtfChars, Tcl_UtfFindFirst, Tcl_UtfFindLast, Tcl_UtfNext, Tcl_UtfPrev, Tcl_UniCharAtIndex, Tcl_UtfAtIndex, Tcl_UtfBackslash - routines for manipulating UTF-8 strings</td>
</tr>
</tbody>
</table>

**SYNOPSIS**

```c
#include <tcl.h>
typedef ... Tcl_UniChar;
int
Tcl_UniCharToUtf(ch, buf)
```
int
Tcl_UtfToUniChar(src, chPtr)
char *
Tcl_UniCharToUtfDString(uniStr, uniLength, dsPtr)
Tcl_UniChar *
Tcl_UtfToUniCharDString(src, length, dsPtr)
int
Tcl_UniCharLen(uniStr)
int
Tcl_UniCharNcmp(ucs, uct, numChars)
int
Tcl_UniCharNcasecmp(ucs, uct, numChars)
int
Tcl_UniCharCaseMatch(uniStr, uniPattern, nocase)
int
Tcl_UtfNcmp(cs, ct, numChars)
int
Tcl_UtfNcasecmp(cs, ct, numChars)
int
Tcl_UtfCharComplete(src, length)
int
Tcl_NumUtfChars(src, length)
const char *
Tcl_UtfFindFirst(src, ch)
const char *
Tcl_UtfFindLast(src, ch)
const char *
Tcl_UtfNext(src)
const char *
Tcl_UtfPrev(src, start)
Tcl_UniChar
Tcl_UniCharAtIndex(src, index)
const char *
Tcl_UtfAtIndex(src, index)
int
Tcl_UtfBackslash(src, readPtr, dst)
<table>
<thead>
<tr>
<th>ARGUMENTS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char *buf (out)</code></td>
<td>Buffer in which the UTF-8 representation of the <code>Tcl_UniChar</code> is stored. At most <code>TCL_UTF_MAX</code> bytes are stored in the buffer.</td>
</tr>
<tr>
<td><code>int ch (in)</code></td>
<td>The <code>Tcl_UniChar</code> to be converted or examined.</td>
</tr>
<tr>
<td><code>Tcl_UniChar *chPtr (out)</code></td>
<td>Filled with the <code>Tcl_UniChar</code> represented by the head of the UTF-8 string.</td>
</tr>
<tr>
<td><code>const char *src (in)</code></td>
<td>Pointer to a UTF-8 string.</td>
</tr>
<tr>
<td><code>const char *cs (in)</code></td>
<td>Pointer to a UTF-8 string.</td>
</tr>
<tr>
<td><code>const char *ct (in)</code></td>
<td>Pointer to a UTF-8 string.</td>
</tr>
<tr>
<td><code>const Tcl_UniChar *uniStr (in)</code></td>
<td>A null-terminated Unicode string.</td>
</tr>
<tr>
<td><code>const Tcl_UniChar *ucs (in)</code></td>
<td>A null-terminated Unicode string.</td>
</tr>
<tr>
<td><code>const Tcl_UniChar *uct (in)</code></td>
<td>A null-terminated Unicode string.</td>
</tr>
<tr>
<td><code>const Tcl_UniChar *uniPattern (in)</code></td>
<td>A null-terminated Unicode string.</td>
</tr>
<tr>
<td><code>int length (in)</code></td>
<td>The length of the UTF-8 string in bytes (not UTF-8 characters). If negative, all</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>int uniLength (in)</td>
<td>The length of the Unicode string in characters. Must be greater than or equal to 0.</td>
</tr>
<tr>
<td>Tcl_DString *dsPtr (in/out)</td>
<td>A pointer to a previously initialized <code>Tcl_DString</code>.</td>
</tr>
<tr>
<td>unsigned long numChars (in)</td>
<td>The number of characters to compare.</td>
</tr>
<tr>
<td>const char *start (in)</td>
<td>Pointer to the beginning of a UTF-8 string.</td>
</tr>
<tr>
<td>int index (in)</td>
<td>The index of a character (not byte) in the UTF-8 string.</td>
</tr>
<tr>
<td>int *readPtr (out)</td>
<td>If non-NULL, filled with the number of bytes in the backslash sequence, including the backslash character.</td>
</tr>
<tr>
<td>char *dst (out)</td>
<td>Buffer in which the bytes represented by the backslash sequence are stored. At most <code>TCL_UTF_MAX</code> bytes are stored in the buffer.</td>
</tr>
<tr>
<td>int nocase (in)</td>
<td>Specifies whether the match should be done case-sensitive (0) or case-</td>
</tr>
</tbody>
</table>
These routines convert between UTF-8 strings and Tcl_UniChars. A Tcl_UniChar is a Unicode character represented as an unsigned, fixed-size quantity. A UTF-8 character is a Unicode character represented as a varying-length sequence of up to TCL_UTF_MAX bytes. A multibyte UTF-8 sequence consists of a lead byte followed by some number of trail bytes.

TCL_UTF_MAX is the maximum number of bytes that it takes to represent one Unicode character in the UTF-8 representation.

Tcl_UniCharToUtf stores the Tcl_UniChar ch as a UTF-8 string in starting at buf. The return value is the number of bytes stored in buf.

Tcl_UtfToUniChar reads one UTF-8 character starting at src and stores it as a Tcl_UniChar in *chPtr. The return value is the number of bytes read from src. The caller must ensure that the source buffer is long enough such that this routine does not run off the end and dereference non-existent or random memory; if the source buffer is known to be null-terminated, this will not happen. If the input is not in proper UTF-8 format, Tcl_UtfToUniChar will store the first byte of src in *chPtr as a Tcl_UniChar between 0x0000 and 0x00ff and return 1.

Tcl_UniCharToUtfDString converts the given Unicode string to UTF-8, storing the result in a previously initialized Tcl_DString. You must specify uniLength, the length of the given Unicode string. The return value is a pointer to the UTF-8 representation of the Unicode string. Storage for the return value is appended to the end of the Tcl_DString.

Tcl_UtfToUniCharDString converts the given UTF-8 string to Unicode, storing the result in the previously initialized Tcl_DString. In the argument length, you may either specify the length of the given UTF-8 string in bytes or “-1”, in which case Tcl_UtfToUniCharDString uses strlen to calculate the length. The return value is a pointer to the
Unicode representation of the UTF-8 string. Storage for the return value is appended to the end of the `Tcl_DString`. The Unicode string is terminated with a Unicode null character.

`Tcl_UniCharLen` corresponds to `strlen` for Unicode characters. It accepts a null-terminated Unicode string and returns the number of Unicode characters (not bytes) in that string.

`Tcl_UniCharNcmp` and `Tcl_UniCharNcasecmp` correspond to `strncmp` and `strncasecmp`, respectively, for Unicode characters. They accept two null-terminated Unicode strings and the number of characters to compare. Both strings are assumed to be at least `numChars` characters long. `Tcl_UniCharNcmp` compares the two strings character-by-character according to the Unicode character ordering. It returns an integer greater than, equal to, or less than 0 if the first string is greater than, equal to, or less than the second string respectively. `Tcl_UniCharNcasecmp` is the Unicode case insensitive version.

`Tcl_UniCharCaseMatch` is the Unicode equivalent to `Tcl_StringCaseMatch`. It accepts a null-terminated Unicode string, a Unicode pattern, and a boolean value specifying whether the match should be case sensitive and returns whether the string matches the pattern.

`Tcl_UtfNcmp` corresponds to `strncmp` for UTF-8 strings. It accepts two null-terminated UTF-8 strings and the number of characters to compare. (Both strings are assumed to be at least `numChars` characters long.) `Tcl_UtfNcmp` compares the two strings character-by-character according to the Unicode character ordering. It returns an integer greater than, equal to, or less than 0 if the first string is greater than, equal to, or less than the second string respectively.

`Tcl_UtfNcasecmp` corresponds to `strncasecmp` for UTF-8 strings. It is similar to `Tcl_UtfNcmp` except comparisons ignore differences in case when comparing upper, lower or title case characters.

`Tcl_UtfCharComplete` returns 1 if the source UTF-8 string `src` of `length`
bytes is long enough to be decoded by `TclUtfToUniChar`, or 0 otherwise. This function does not guarantee that the UTF-8 string is properly formed. This routine is used by procedures that are operating on a byte at a time and need to know if a full `Tcl_UniChar` has been seen.

`Tcl_NumUtfChars` corresponds to `strlen` for UTF-8 strings. It returns the number of `Tcl_UniChars` that are represented by the UTF-8 string `src`. The length of the source string is `length` bytes. If the length is negative, all bytes up to the first null byte are used.

`Tcl_UtfFindFirst` corresponds to `strchr` for UTF-8 strings. It returns a pointer to the first occurrence of the `Tcl_UniChar` `ch` in the null-terminated UTF-8 string `src`. The null terminator is considered part of the UTF-8 string.

`Tcl_UtfFindLast` corresponds to `strrchr` for UTF-8 strings. It returns a pointer to the last occurrence of the `Tcl_UniChar` `ch` in the null-terminated UTF-8 string `src`. The null terminator is considered part of the UTF-8 string.

Given `src`, a pointer to some location in a UTF-8 string, `Tcl_UtfNext` returns a pointer to the next UTF-8 character in the string. The caller must not ask for the next character after the last character in the string if the string is not terminated by a null character.

Given `src`, a pointer to some location in a UTF-8 string (or to a null byte immediately following such a string), `Tcl_UtfPrev` returns a pointer to the closest preceding byte that starts a UTF-8 character. This function will not back up to a position before `start`, the start of the UTF-8 string. If `src` was already at `start`, the return value will be `start`.

`Tcl_UniCharAtIndex` corresponds to a C string array dereference or the Pascal `Ord()` function. It returns the `Tcl_UniChar` represented at the specified character (not byte) `index` in the UTF-8 string `src`. The source string must contain at least `index` characters. Behavior is undefined if a negative `index` is given.
**Tcl_UtfAtIndex** returns a pointer to the specified character (not byte) index in the UTF-8 string src. The source string must contain at least index characters. This is equivalent to calling **Tcl_UtfNext** index times. If a negative index is given, the return pointer points to the first character in the source string.

**Tcl_UtfBackslash** is a utility procedure used by several of the Tcl commands. It parses a backslash sequence and stores the properly formed UTF-8 character represented by the backslash sequence in the output buffer dst. At most TCL_UTF_MAX bytes are stored in the buffer. **Tcl_UtfBackslash** modifies *readPtr to contain the number of bytes in the backslash sequence, including the backslash character. The return value is the number of bytes stored in the output buffer.

See the Tcl manual entry for information on the valid backslash sequences. All of the sequences described in the Tcl manual entry are supported by **Tcl_UtfBackslash**.

**KEYWORDS**

utf, unicode, backslash

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997 Sun Microsystems, Inc.
NAME
Tcl_SetVar2Ex, Tcl_SetVar, Tcl_SetVar2, Tcl_ObjSetVar2,
Tcl_GetVar2Ex, Tcl_GetVar, Tcl_GetVar2, Tcl_ObjGetVar2,
Tcl_UnsetVar, Tcl_UnsetVar2 - manipulate Tcl variables

SYNOPSIS
#include <tcl.h>
Tcl_Obj *
Tcl_SetVar2Ex(interp, name1, name2, newValuePtr, flags)
const char *
Tcl_SetVar(interp, varName, newValue, flags)
const char *
Tcl_SetVar2(interp, name1, name2, newValue, flags)
Tcl_Obj *
Tcl_ObjSetVar2(interp, part1Ptr, part2Ptr, newValuePtr, flags)
Tcl_Obj *
Tcl_GetVar2Ex(interp, name1, name2, flags)
const char *
Tcl_GetVar(interp, varName, flags)
const char *
Tcl_GetVar2(interp, name1, name2, flags)
Tcl_Obj *
Tcl_ObjGetVar2(interp, part1Ptr, part2Ptr, flags)
int
Tcl_UnsetVar(interp, varName, flags)
int
Tcl_UnsetVar2(interp, name1, name2, flags)

ARGUMENTS

DESCRIPTION
TCL_GLOBAL_ONLY
TCL_NAMESPACE_ONLY
TCL_LEAVE_ERR_MSG
**NAME**

Tcl_SetVar2Ex, Tcl_SetVar, Tcl_SetVar2, Tcl_ObjSetVar2, Tcl_GetVar2Ex, Tcl_GetVar, Tcl_GetVar2, Tcl_ObjGetVar2, Tcl_UnsetVar, Tcl_UnsetVar2 - manipulate Tcl variables

**SYNOPSIS**

```c
#include <tcl.h>
Tcl_Obj *
Tcl_SetVar2Ex(interp, name1, name2, newValuePtr, flags)
const char *
Tcl_SetVar(interp, varName, newValue, flags)
const char *
Tcl_SetVar2(interp, name1, name2, newValue, flags)
Tcl_Obj *
Tcl_ObjSetVar2(interp, part1Ptr, part2Ptr, newValuePtr, flags)
Tcl_Obj *
Tcl_GetVar2Ex(interp, name1, name2, flags)
const char *
Tcl_GetVar(interp, varName, flags)
const char *
Tcl_GetVar2(interp, name1, name2, flags)
Tcl_Obj *
Tcl_ObjGetVar2(interp, part1Ptr, part2Ptr, flags)
int
Tcl_UnsetVar(interp, varName, flags)
int
Tcl_UnsetVar2(interp, name1, name2, flags)
```

**ARGUMENTS**

Tcl_Interp *interp (in)  Interpreter containing
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const char *name1</td>
<td>Contains the name of an array variable (if name2 is non-NULL) or (if name2 is NULL) either the name of a scalar variable or a complete name including both variable name and index. May include :: namespace qualifiers to specify a variable in a particular namespace.</td>
</tr>
<tr>
<td>const char *name2</td>
<td>If non-NULL, gives name of element within array; in this case name1 must refer to an array variable.</td>
</tr>
<tr>
<td>Tcl_Obj *newValuePtr</td>
<td>Points to a Tcl object containing the new value for the variable.</td>
</tr>
<tr>
<td>int flags</td>
<td>OR-ed combination of bits providing additional information. See below for valid values.</td>
</tr>
<tr>
<td>const char *varName</td>
<td>Name of variable. May include :: namespace qualifiers to specify a variable in a particular namespace. May refer to a scalar variable or an element of an array.</td>
</tr>
<tr>
<td>const char *newValue</td>
<td>New value for variable,</td>
</tr>
</tbody>
</table>
specified as a null-terminated string. A copy of this value is stored in the variable.

Tcl_Obj *part1Ptr (in) Points to a Tcl object containing the variable's name. The name may include a series of :: namespace qualifiers to specify a variable in a particular namespace. May refer to a scalar variable or an element of an array variable.

Tcl_Obj *part2Ptr (in) If non-NULL, points to an object containing the name of an element within an array and part1Ptr must refer to an array variable.

DESCRIPTION

These procedures are used to create, modify, read, and delete Tcl variables from C code.

Tcl_SetVar2Ex, Tcl_SetVar, Tcl_SetVar2, and Tcl_ObjSetVar2 will create a new variable or modify an existing one. These procedures set the given variable to the value given by newValuePtr or newValue and return a pointer to the variable's new value, which is stored in Tcl's variable structure. Tcl_SetVar2Ex and Tcl_ObjSetVar2 take the new value as a Tcl_Obj and return a pointer to a Tcl_Obj. Tcl_SetVar and Tcl_SetVar2 take the new value as a string and return a string; they are usually less efficient than Tcl_ObjSetVar2. Note that the return value may be different than the newValuePtr or newValue argument, due to
modifications made by write traces. If an error occurs in setting the variable (e.g. an array variable is referenced without giving an index into the array) NULL is returned and an error message is left in interp's result if the TCL_LEAVE_ERR_MSG flag bit is set.

Tcl_GetVar2Ex, Tcl_GetVar, Tcl_GetVar2, and Tcl_ObjGetVar2 return the current value of a variable. The arguments to these procedures are treated in the same way as the arguments to the procedures described above. Under normal circumstances, the return value is a pointer to the variable's value. For Tcl_GetVar2Ex and Tcl_ObjGetVar2 the value is returned as a pointer to a Tcl_Obj. For Tcl_GetVar and Tcl_GetVar2 the value is returned as a string; this is usually less efficient, so Tcl_GetVar2Ex or Tcl_ObjGetVar2 are preferred. If an error occurs while reading the variable (e.g. the variable does not exist or an array element is specified for a scalar variable), then NULL is returned and an error message is left in interp's result if the TCL_LEAVE_ERR_MSG flag bit is set.

Tcl_UnsetVar and Tcl_UnsetVar2 may be used to remove a variable, so that future attempts to read the variable will return an error. The arguments to these procedures are treated in the same way as the arguments to the procedures above. If the variable is successfully removed then TCL_OK is returned. If the variable cannot be removed because it does not exist then TCL_ERROR is returned and an error message is left in interp's result if the TCL_LEAVE_ERR_MSG flag bit is set. If an array element is specified, the given element is removed but the array remains. If an array name is specified without an index, then the entire array is removed.

The name of a variable may be specified to these procedures in four ways:

[1] If Tcl_SetVar, Tcl_GetVar, or Tcl_UnsetVar is invoked, the variable name is given as a single string, varName. If varName contains an open parenthesis and ends with a close parenthesis, then the value between the parentheses is treated as an index (which can have any string value) and the characters before the
first open parenthesis are treated as the name of an array variable. If \textit{varName} does not have parentheses as described above, then the entire string is treated as the name of a scalar variable.

[2] If the \textit{name1} and \textit{name2} arguments are provided and \textit{name2} is non-NULL, then an array element is specified and the array name and index have already been separated by the caller: \textit{name1} contains the name and \textit{name2} contains the index. An error is generated if \textit{name1} contains an open parenthesis and ends with a close parenthesis (array element) and \textit{name2} is non-NULL.

[3] If \textit{name2} is NULL, \textit{name1} is treated just like \textit{varName} in case [1] above (it can be either a scalar or an array element variable name).

The \textit{flags} argument may be used to specify any of several options to the procedures. It consists of an OR-ed combination of the following bits.

\textbf{TCL\_GLOBAL\_ONLY} 
Under normal circumstances the procedures look up variables as follows. If a procedure call is active in \textit{interp}, the variable is looked up at the current level of procedure call. Otherwise, the variable is looked up first in the current namespace, then in the global namespace. However, if this bit is set in \textit{flags} then the variable is looked up only in the global namespace even if there is a procedure call active. If both \textbf{TCL\_GLOBAL\_ONLY} and \textbf{TCL\_NAMESPACE\_ONLY} are given, \textbf{TCL\_GLOBAL\_ONLY} is ignored.

\textbf{TCL\_NAMESPACE\_ONLY} 
If this bit is set in \textit{flags} then the variable is looked up only in the current namespace; if a procedure is active its variables are ignored, and the global namespace is also ignored unless it is the current namespace.

\textbf{TCL\_LEAVE\_ERR\_MSG}
If an error is returned and this bit is set in \textit{flags}, then an error message will be left in the interpreter's result, where it can be retrieved with \texttt{Tcl\_GetObjResult} or \texttt{Tcl\_GetStringResult}. If this flag bit is not set then no error message is left and the interpreter's result will not be modified.

\textbf{TCL\_APPEND\_VALUE}

If this bit is set then \texttt{newValuePtr} or \texttt{newValue} is appended to the current value instead of replacing it. If the variable is currently undefined, then the bit is ignored. This bit is only used by the \texttt{Tcl\_Set\*} procedures.

\textbf{TCL\_LIST\_ELEMENT}

If this bit is set, then \texttt{newValue} is converted to a valid Tcl list element before setting (or appending to) the variable. A separator space is appended before the new list element unless the list element is going to be the first element in a list or sublist (i.e. the variable's current value is empty, or contains the single character “{”, or ends in “ }”). When appending, the original value of the variable must also be a valid list, so that the operation is the appending of a new list element onto a list.

\texttt{Tcl\_GetVar} and \texttt{Tcl\_GetVar2} return the current value of a variable. The arguments to these procedures are treated in the same way as the arguments to \texttt{Tcl\_SetVar} and \texttt{Tcl\_SetVar2}. Under normal circumstances, the return value is a pointer to the variable's value (which is stored in Tcl's variable structure and will not change before the next call to \texttt{Tcl\_SetVar} or \texttt{Tcl\_SetVar2}). \texttt{Tcl\_GetVar} and \texttt{Tcl\_GetVar2} use the flag bits \texttt{TCL\_GLOBAL\_ONLY} and \texttt{TCL\_LEAVE\_ERR\_MSG}, both of which have the same meaning as for \texttt{Tcl\_SetVar}. If an error occurs in reading the variable (e.g. the variable does not exist or an array element is specified for a scalar variable), then NULL is returned.

\texttt{Tcl\_UnsetVar} and \texttt{Tcl\_UnsetVar2} may be used to remove a variable, so that future calls to \texttt{Tcl\_GetVar} or \texttt{Tcl\_GetVar2} for the variable will return an error. The arguments to these procedures are treated in the same way as the arguments to \texttt{Tcl\_GetVar} and \texttt{Tcl\_GetVar2}. If the variable is successfully removed then \texttt{TCL\_OK} is returned. If the
variable cannot be removed because it does not exist then `TCL_ERROR` is returned. If an array element is specified, the given element is removed but the array remains. If an array name is specified without an index, then the entire array is removed.

**SEE ALSO**

`Tcl_GetObjResult`, `Tcl_GetStringResult`, `Tcl_TraceVar`  

**KEYWORDS**

array, get variable, interpreter, object, scalar, set, unset, variable

Copyright © 1995-1997 Roger E. Critchlow Jr.  
Copyright © 1989-1993 The Regents of the University of California.  
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME
Tcl_NewStringObj, Tcl_NewUnicodeObj, Tcl_SetStringObj,
Tcl_SetUnicodeObj, Tcl_GetStringFromObj, Tcl_GetString,
Tcl_GetUnicodeFromObj, Tcl_GetUnicode, Tcl_GetUniChar,
Tcl_GetCharLength, Tcl_GetRange, Tcl_AppendToObj,
Tcl_AppendUnicodeToObj, Tcl_AppendObjToObj,
Tcl_AppendStringsToObj, Tcl_AppendStringsToObjVA,
Tcl_AppendLimitedToObj, Tcl_Format,
Tcl_AppendFormatToObj, Tcl_ObjPrintf,
Tcl_AppendPrintfToObj, Tcl_SetObjLength,
Tcl_AttemptSetObjLength, Tcl_ConcatObj - manipulate Tcl
objects as strings

SYNOPSIS
#include <tcl.h>
Tcl_Obj *
Tcl_NewStringObj(bytes, length)
Tcl_Obj *
Tcl_NewUnicodeObj(unicode, numChars)
void
Tcl_SetStringObj(objPtr, bytes, length)
void
Tcl_SetUnicodeObj(objPtr, unicode, numChars)
char *
Tcl_GetStringFromObj(objPtr, lengthPtr)
char *
Tcl_GetString(objPtr)
Tcl_UniChar *
Tcl_GetUnicodeFromObj(objPtr, lengthPtr)
Tcl_UniChar *
Tcl_GetUnicode(objPtr)
Tcl_UniChar
Tcl_GetUniChar(objPtr, index)
int
Tcl_GetCharLength(objPtr)
Tcl_Obj *
Tcl_GetRange(objPtr, first, last)
void
Tcl_AppendToObj(objPtr, bytes, length)
void
Tcl_AppendUnicodeToObj(objPtr, unicode, numChars)
void
Tcl_AppendObjToObj(objPtr, appendObjPtr)
void
Tcl_AppendStringsToObj(objPtr, string, string, ... (char *) NULL)
void
Tcl_AppendStringsToObjVA(objPtr, argList)
void
Tcl_AppendLimitedToObj(objPtr, bytes, length, limit, ellipsis)
Tcl_Obj *
Tcl_Format(interp, format, objc, objv)
int
Tcl_AppendFormatToObj(interp, objPtr, format, objc, objv)
Tcl_Obj *
Tcl_ObjPrintf(format, ...)
int
Tcl_AppendPrintfToObj(objPtr, format, ...)
void
Tcl_SetObjLength(objPtr, newLength)
int
Tcl_AttemptSetObjLength(objPtr, newLength)
Tcl_Obj *
Tcl_ConcatObj(objc, objv)
NAME

Tcl_NewStringObj, Tcl_NewUnicodeObj, Tcl_SetStringObj,
Tcl_SetUnicodeObj, Tcl_GetStringFromObj, Tcl_GetString,
Tcl_GetUnicodeFromObj, Tcl_GetUnicode, Tcl_GetUniChar,
Tcl_GetCharLength, Tcl_GetRange, Tcl_AppendToObj,
Tcl_AppendUnicodeToObj, Tcl_AppendObjToObj,
Tcl_AppendStringsToObj, Tcl_AppendStringsToObjVA,
Tcl_AppendLimitedToObj, Tcl_Format, Tcl_AppendFormatToObj,
Tcl_ObjPrintf, Tcl_AppendPrintfToObj, Tcl_SetObjLength,
Tcl_AttemptSetObjLength, Tcl_ConcatObj - manipulate Tcl objects as strings

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_NewStringObj(bytes, length)
Tcl_Obj *
Tcl_NewUnicodeObj(unicode, numChars)
void
Tcl_SetStringObj(objPtr, bytes, length)
void
Tcl_SetUnicodeObj(objPtr, unicode, numChars)
char *
Tcl_GetStringFromObj(objPtr, lengthPtr)
char *
Tcl_GetString(objPtr)
Tcl_UniChar *
Tcl_GetUnicodeFromObj(objPtr, lengthPtr)
Tcl_UniChar *
Tcl_GetUnicode(objPtr)
Tcl_UniChar
Tcl_GetUniChar(objPtr, index)
int
Tcl_GetCharLength(objPtr)
Tcl_Obj *
const char *bytes (in)  Points to the first byte of an array of UTF-8-encoded bytes used to set or append to a string object. This byte array may contain embedded null characters unless
numChars is negative. (Applications needing null bytes should represent them as the two-byte sequence \170\600, use *Tcl.getExternalToUtf* to convert, or *Tcl.NewByteArrayObj* if the string is a collection of uninterpreted bytes.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>int length (in)</strong></td>
<td>The number of bytes to copy from bytes when initializing, setting, or appending to a string object. If negative, all bytes up to the first null are used.</td>
</tr>
<tr>
<td>*<em>const Tcl_UniChar <em>unicode (in)</em></em></td>
<td>Points to the first byte of an array of Unicode characters used to set or append to a string object. This byte array may contain embedded null characters unless numChars is negative.</td>
</tr>
<tr>
<td><strong>int numChars (in)</strong></td>
<td>The number of Unicode characters to copy from unicode when initializing, setting, or appending to a string object. If negative, all characters up to the first null character are used.</td>
</tr>
<tr>
<td><strong>int index (in)</strong></td>
<td>The index of the Unicode</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>int first (in)</strong></td>
<td>The index of the first Unicode character in the Unicode range to be returned as a new object.</td>
</tr>
<tr>
<td><strong>int last (in)</strong></td>
<td>The index of the last Unicode character in the Unicode range to be returned as a new object.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>objPtr (in/out)</em></em></td>
<td>Points to an object to manipulate.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>appendObjPtr (in)</em></em></td>
<td>The object to append to objPtr in Tcl_AppendObjToObj.</td>
</tr>
<tr>
<td>*<em>int <em>lengthPtr (out)</em></em></td>
<td>If non-NULL, the location where Tcl_GetStringFromObj will store the length of an object's string representation.</td>
</tr>
<tr>
<td>*<em>const char <em>string (in)</em></em></td>
<td>Null-terminated string value to append to objPtr.</td>
</tr>
<tr>
<td><strong>va_list argList (in)</strong></td>
<td>An argument list which must have been initialised using va_start, and cleared using va_end.</td>
</tr>
<tr>
<td><strong>int limit (in)</strong></td>
<td>Maximum number of bytes to be appended.</td>
</tr>
</tbody>
</table>
const char *ellipsis (in)  Suffix to append when the limit leads to string truncation. If NULL is passed then the suffix "..." is used.

const char *format (in)  Format control string including % conversion specifiers.

int objc (in)  The number of elements to format or concatenate.

Tcl_Obj *objv[] (in)  The array of objects to format or concatenate.

int newLength (in)  New length for the string value of objPtr, not including the final null character.

**DESCRIPTION**

The procedures described in this manual entry allow Tcl objects to be manipulated as string values. They use the internal representation of the object to store additional information to make the string manipulations more efficient. In particular, they make a series of append operations efficient by allocating extra storage space for the string so that it does not have to be copied for each append. Also, indexing and length computations are optimized because the Unicode string representation is calculated and cached as needed. When using the `Tcl_Append*` family of functions where the interpreter's result is the object being appended to, it is important to call `Tcl_ResetResult` first to ensure you are not unintentionally appending to existing data in the result object.
Tcl_NewStringObj and Tcl_SetStringObj create a new object or modify an existing object to hold a copy of the string given by bytes and length. Tcl_NewUnicodeObj and Tcl_SetUnicodeObj create a new object or modify an existing object to hold a copy of the Unicode string given by unicode and numChars. Tcl_NewStringObj and Tcl_NewUnicodeObj return a pointer to a newly created object with reference count zero. All four procedures set the object to hold a copy of the specified string. Tcl_SetStringObj and Tcl_SetUnicodeObj free any old string representation as well as any old internal representation of the object.

Tcl_GetStringFromObj and Tcl_GetString return an object's string representation. This is given by the returned byte pointer and (for Tcl_GetStringFromObj) length, which is stored in lengthPtr if it is non-NULL. If the object's UTF string representation is invalid (its byte pointer is NULL), the string representation is regenerated from the object's internal representation. The storage referenced by the returned byte pointer is owned by the object manager. It is passed back as a writable pointer so that extension author creating their own Tcl_ObjType will be able to modify the string representation within the Tcl_UpdateStringProc of their Tcl_ObjType. Except for that limited purpose, the pointer returned by Tcl_GetStringFromObj or Tcl_GetString should be treated as read-only. It is recommended that this pointer be assigned to a (const char *) variable. Even in the limited situations where writing to this pointer is acceptable, one should take care to respect the copy-on-write semantics required by Tcl_Obj's, with appropriate calls to Tcl_IsShared and Tcl_DuplicateObj prior to any in-place modification of the string representation. The procedure Tcl_GetString is used in the common case where the caller does not need the length of the string representation.

Tcl_GetUnicodeFromObj and Tcl_GetUnicode return an object's value as a Unicode string. This is given by the returned pointer and (for Tcl_GetUnicodeFromObj) length, which is stored in lengthPtr if it is non-NULL. The storage referenced by the returned byte pointer is owned by the object manager and should not be modified by the caller. The procedure Tcl_GetUnicode is used in the common case where the
caller does not need the length of the unicode string representation.

**Tcl_GetUniChar** returns the index' th character in the object's Unicode representation.

**Tcl_GetRange** returns a newly created object comprised of the characters between first and last (inclusive) in the object's Unicode representation. If the object's Unicode representation is invalid, the Unicode representation is regenerated from the object's string representation.

**Tcl_GetCharLength** returns the number of characters (as opposed to bytes) in the string object.

**Tcl_AppendToObj** appends the data given by bytes and length to the string representation of the object specified by objPtr. If the object has an invalid string representation, then an attempt is made to convert bytes is to the Unicode format. If the conversion is successful, then the converted form of bytes is appended to the object's Unicode representation. Otherwise, the object's Unicode representation is invalidated and converted to the UTF format, and bytes is appended to the object's new string representation.

**Tcl_AppendUnicodeToObj** appends the Unicode string given by unicode and numChars to the object specified by objPtr. If the object has an invalid Unicode representation, then unicode is converted to the UTF format and appended to the object's string representation. Appends are optimized to handle repeated appends relatively efficiently (it overallocates the string or Unicode space to avoid repeated reallocations and copies of object's string value).

**Tcl_AppendObjToObj** is similar to Tcl_AppendToObj, but it appends the string or Unicode value (whichever exists and is best suited to be appended to objPtr) of appendObjPtr to objPtr.

**Tcl_AppendStringsToObj** is similar to Tcl_AppendToObj except that it can be passed more than one value to append and each value must be a null-terminated string (i.e. none of the values may contain internal null
Any number of *string* arguments may be provided, but the last argument must be a NULL pointer to indicate the end of the list.

**Tcl_AppendStringsToObjVA** is the same as **Tcl_AppendStringsToObj** except that instead of taking a variable number of arguments it takes an argument list.

**Tcl_AppendLimitedToObj** is similar to **Tcl_AppendToObj** except that it imposes a limit on how many bytes are appended. This can be handy when the string to be appended might be very large, but the value being constructed should not be allowed to grow without bound. A common usage is when constructing an error message, where the end result should be kept short enough to be read. Bytes from *bytes* are appended to *objPtr*, but no more than *limit* bytes total are to be appended. If the limit prevents all *length* bytes that are available from being appended, then the appending is done so that the last bytes appended are from the string *ellipsis*. This allows for an indication of the truncation to be left in the string. When *length* is -1, all bytes up to the first zero byte are appended, subject to the limit. When *ellipsis* is NULL, the default string ... is used. When *ellipsis* is non-NULL, it must point to a zero-byte-terminated string in Tcl's internal UTF encoding. The number of bytes appended can be less than the lesser of *length* and *limit* when appending fewer bytes is necessary to append only whole multi-byte characters.

**Tcl_Format** is the C-level interface to the engine of the *format* command. The actual command procedure for *format* is little more than

```
Tcl_Format(interp, Tcl_GetString(objv[1]), objc-2, objc+2);
```

The *objc* *Tcl_Obj* values in *objv* are formatted into a string according to the conversion specification in *format* argument, following the documentation for the *format* command. The resulting formatted string is converted to a new *Tcl_Obj* with refcount of zero and returned. If some error happens during production of the formatted string, NULL is
returned, and an error message is recorded in interp, if interp is non-NULL.

Tcl_AppendFormatToObj is an appending alternative form of Tcl_Format with functionality equivalent to

```c
Tcl_Obj *newPtr = Tcl_Format(interp, format, objc, objv);
if (newPtr == NULL) return TCL_ERROR;
Tcl_AppendObjToObj(objPtr, newPtr);
return TCL_OK;
```

but with greater convenience and efficiency when the appending functionality is needed.

Tcl_ObjPrintf serves as a replacement for the common sequence

```c
char buf[SOME_SUITABLE_LENGTH];
sprintf(buf, format, ...);
Tcl_NewStringObj(buf, -1);
```

but with greater convenience and no need to determine SOME_SUITABLE_LENGTH. The formatting is done with the same core formatting engine used by Tcl_Format. This means the set of supported conversion specifiers is that of the format command and not that of the sprintf routine where the two sets differ. When a conversion specifier passed to Tcl_ObjPrintf includes a precision, the value is taken as a number of bytes, as sprintf does, and not as a number of characters, as format does. This is done on the assumption that C code is more likely to know how many bytes it is passing around than the number of encoded characters those bytes happen to represent. The variable number of arguments passed in should be of the types that would be suitable for passing to sprintf. Note in this example usage, x is of type long.
long x = 5;
Tcl_Obj *objPtr = Tcl_ObjPrintf("Value is %d", x);

If the value of format contains internal inconsistencies or invalid specifier formats, the formatted string result produced by Tcl_ObjPrintf will be an error message describing the error.

Tcl_AppendPrintfToObj is an appending alternative form of Tcl_ObjPrintf with functionality equivalent to

```
Tcl_AppendObjToObj(objPtr, Tcl_ObjPrintf(format, ...));
```

but with greater convenience and efficiency when the appending functionality is needed.

The Tcl_SetObjLength procedure changes the length of the string value of its objPtr argument. If the newLength argument is greater than the space allocated for the object's string, then the string space is reallocated and the old value is copied to the new space; the bytes between the old length of the string and the new length may have arbitrary values. If the newLength argument is less than the current length of the object's string, with objPtr->length is reduced without reallocating the string space; the original allocated size for the string is recorded in the object, so that the string length can be enlarged in a subsequent call to Tcl_SetObjLength without reallocating storage. In all cases Tcl_SetObjLength leaves a null character at objPtr->bytes[newLength].

Tcl_AttemptSetObjLength is identical in function to Tcl_SetObjLength except that if sufficient memory to satisfy the request cannot be allocated, it does not cause the Tcl interpreter to panic. Thus, if newLength is greater than the space allocated for the object's string, and there is not enough memory available to satisfy the request, Tcl_AttemptSetObjLength will take no action and return 0 to
indicate failure. If there is enough memory to satisfy the request, \texttt{Tcl\_AttemptSetObjLength} behaves just like \texttt{Tcl\_SetObjLength} and returns 1 to indicate success.

The \texttt{Tcl\_ConcatObj} function returns a new string object whose value is the space-separated concatenation of the string representations of all of the objects in the \texttt{objv} array. \texttt{Tcl\_ConcatObj} eliminates leading and trailing white space as it copies the string representations of the \texttt{objv} array to the result. If an element of the \texttt{objv} array consists of nothing but white space, then that object is ignored entirely. This white-space removal was added to make the output of the \texttt{concat} command cleaner-looking. \texttt{Tcl\_ConcatObj} returns a pointer to a newly-created object whose ref count is zero.

\begin{center}
\textbf{SEE ALSO}
\end{center}
\begin{itemize}
\item \texttt{Tcl\_NewObj}, \texttt{Tcl\_IncrRefCount}, \texttt{Tcl\_DecrRefCount}, \texttt{format}, \texttt{sprintf}
\end{itemize}

\begin{center}
\textbf{KEYWORDS}
\end{center}
\begin{itemize}
\item append, internal representation, object, object type, string object, string type, string representation, \texttt{concat}, \texttt{concatenate}, \texttt{unicode}
\end{itemize}
**NAME**

Tcl_OpenFileChannel, Tcl_OpenCommandChannel,
Tcl_MakeFileChannel, Tcl_GetChannel,
Tcl_GetChannelNames, Tcl_GetChannelNamesEx,
Tcl_RegisterChannel, Tcl_UnregisterChannel,
Tcl_DetachChannel, Tcl_IsStandardChannel, Tcl_Close,
Tcl_ReadChars, Tcl_Read, Tcl_GetObj, Tcl_Gets,
Tcl_WriteObj, Tcl_WriteChars, Tcl_Write, Tcl_Flush, Tcl_Sek,
Tcl_Tell, Tcl_TruncateChannel, Tcl_GetChannelOption,
Tcl_SetChannelOption, Tcl_Eof, Tcl_InputBlocked,
Tcl_InputBuffered, Tcl_OutputBuffered, Tcl_Ungets,
Tcl_ReadRaw, Tcl_WriteRaw - buffered I/O facilities using channels

**SYNOPSIS**

#include <tcl.h>
Tcl_Channel
Tcl_OpenFileChannel(*interp, fileName, mode, permissions*)
Tcl_Channel
Tcl_OpenCommandChannel(*interp, argc, argv, flags*)
Tcl_Channel
Tcl_MakeFileChannel(*handle, readOrWrite*)
Tcl_Channel
Tcl_GetChannel(*interp, channelName, modePtr*)
int
Tcl_GetChannelNames(*interp*)
int
Tcl_GetChannelNamesEx(*interp, pattern*)
void
Tcl_RegisterChannel(*interp, channel*)
int
Tcl_UnregisterChannel(*interp, channel*)
int Tcl_DetachChannel(interp, channel)
int Tcl_IsStandardChannel(channel)
int Tcl_Close(interp, channel)
int Tcl_ReadChars(channel, readObjPtr, charsToRead, appendFlag)
int Tcl_Read(channel, readBuf, bytesToRead)
int Tcl_GetsObj(channel, lineObjPtr)
int Tcl_Gets(channel, lineRead)
int Tcl_Ungets(channel, input, inputLen, addAtEnd)
int Tcl_WriteObj(channel, writeObjPtr)
int Tcl_WriteChars(channel, charBuf, bytesToWrite)
int Tcl_Write(channel, byteBuf, bytesToWrite)
int Tcl_ReadRaw(channel, readBuf, bytesToRead)
int Tcl_WriteRaw(channel, byteBuf, bytesToWrite)
int Tcl_Eof(channel)
int Tcl_Flush(channel)
int Tcl_InputBlocked(channel)
int Tcl_InputBuffered(channel)
int Tcl_OutputBuffered(channel)
Tcl_WideInt

Tcl_Seek(channel, offset, seekMode)
Tcl_WideInt

Tcl_Tell(channel)
int

Tcl_TruncateChannel(channel, length)
int

Tcl_GetChannelOption(interp, channel, optionName, optionValue)
int

Tcl_SetChannelOption(interp, channel, optionName, newValue)

ARGUMENTS

DESCRIPTION

TCL_OPENFILECHANNEL
TCL_OPENCOMMANDCHANNEL
TCL_MAKEFILECHANNEL
TCL_GETCHANNEL
TCL_REGISTERCHANNEL
TCL_UNREGISTERCHANNEL
TCL_DETACHCHANNEL
TCL_ISSTANDARDCHANNEL
TCL_CLOSE
TCL_READCHARS AND TCL_READ
TCL_GETSOBJ AND TCL_GETS
TCL_UNGETS
TCL_WRITECHARS, TCL_WRITEOBJ, AND TCL_WRITE
TCL_FLUSH
TCL_SEEK
TCL_TELL
TCL_TRUNCATECHANNEL
TCL_GETCHANNELOPTION
TCL_SETCHANNELOPTION
TCL_EOF
TCL_INPUTBLOCKED
TCL_INPUTBUFFERED
TCL_OUTPUTBUFFERED
PLATFORM ISSUES
SEE ALSO
KEYWORDS

NAME

Tcl_OpenFileChannel, Tcl_OpenCommandChannel,
Tcl_MakeFileChannel, Tcl_GetChannel, Tcl_GetChannelNames,
Tcl_GetChannelNamesEx, Tcl_RegisterChannel,
Tcl_UnregisterChannel, Tcl_DetachChannel, Tcl_IsStandardChannel,
Tcl_Close, Tcl_ReadChars, Tcl_Read, Tcl_GetObj, Tcl_Get,
Tcl_WriteObj, Tcl_WriteChars, Tcl_Write, Tcl.Flush, Tcl.Seek, Tcl_Tell,
Tcl_TruncateChannel, Tcl_GetChannelOption, Tcl_SetChannelOption,
Tcl_Eof, Tcl_InputBlocked, Tcl_InputBuffered, Tcl_OutputBuffered,
Tcl_Ungets, Tcl_ReadRaw, Tcl_WriteRaw - buffered I/O facilities using
channels

SYNOPSIS

#include <tcl.h>
Tcl_Channel
Tcl_OpenFileChannel(interp, fileName, mode, permissions)
Tcl_Channel
Tcl_OpenCommandChannel(interp, argc, argv, flags)
Tcl_Channel
Tcl_MakeFileChannel(handle, readOrWrite)
Tcl_Channel
Tcl_GetChannel(interp, channelName, modePtr)
int
Tcl_GetChannelNames(interp)
int
Tcl_GetChannelNamesEx(interp, pattern)
void
Tcl_RegisterChannel(interp, channel)
int
Tcl_UnregisterChannel(interp, channel)
int
Tcl_DetachChannel(interp, channel)
int Tcl_IsStandardChannel(channel)
int Tcl_Close(interp, channel)
int Tcl_ReadChars(channel, readObjPtr, charsToRead, appendFlag)
int Tcl_Read(channel, readBuf, bytesToRead)
int Tcl_GetsObj(channel, lineObjPtr)
int Tcl_Gets(channel, lineRead)
int Tcl_Ungets(channel, input, inputLen, addAtEnd)
int Tcl_WriteObj(channel, writeObjPtr)
int Tcl_WriteChars(channel, charBuf, bytesToWrite)
int Tcl_Write(channel, byteBuf, bytesToWrite)
int Tcl_ReadRaw(channel, readBuf, bytesToRead)
int Tcl_WriteRaw(channel, byteBuf, bytesToWrite)
int Tcl_Eof(channel)
int Tcl_Flush(channel)
int Tcl_InputBlocked(channel)
int Tcl_InputBuffered(channel)
int Tcl_OutputBuffered(channel)
Tcl_WideInt
Tcl_Seek(channel, offset, seekMode)
Tcl_WideInt
Tcl_Tell(channel)
int
Tcl_TruncateChannel(channel, length)
int
Tcl_GetChannelOption(interp, channel, optionName, optionValue)
int
Tcl_SetChannelOption(interp, channel, optionName, newValue)

<table>
<thead>
<tr>
<th>ARGUMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp</em> (in)</td>
</tr>
<tr>
<td>const char *fileName (in)</td>
</tr>
<tr>
<td>const char *mode (in)</td>
</tr>
<tr>
<td>int permissions (in)</td>
</tr>
<tr>
<td>int argc (in)</td>
</tr>
<tr>
<td>const char **argv (in)</td>
</tr>
</tbody>
</table>
**exec** command.

**int flags (in)**

Specifies the disposition of the stdio handles in pipeline: OR-ed combination of TCL_STDIN, TCL_STDOUT, TCL_STDERR, and TCL_ENFORCE_MODE. If TCL_STDIN is set, stdin for the first child in the pipe is the pipe channel, otherwise it is the same as the standard input of the invoking process; likewise for TCL_STDOUT and TCL_STDERR. If TCL_ENFORCE_MODE is not set, then the pipe can redirect stdio handles to override the stdio handles for which TCL_STDIN, TCL_STDOUT and TCL_STDERR have been set. If it is set, then such redirections cause an error.

**ClientData handle (in)**

Operating system specific handle for I/O to a file. For Unix this is a file descriptor, for Windows it is a HANDLE.

**int readOrWrite (in)**

OR-ed combination of TCL_READABLE and
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const char *channelName (in)</td>
<td>The name of the channel.</td>
</tr>
<tr>
<td>int *modePtr (out)</td>
<td>Points at an integer variable that will receive an OR-ed combination of TCL_READABLE and TCL_WRITABLE denoting whether the channel is open for reading and writing.</td>
</tr>
<tr>
<td>const char *pattern (in)</td>
<td>The pattern to match on, passed to Tcl_StringMatch, or NULL.</td>
</tr>
<tr>
<td>Tcl_Channel channel (in)</td>
<td>A Tcl channel for input or output. Must have been the return value from a procedure such as Tcl_OpenFileChannel.</td>
</tr>
<tr>
<td>Tcl_Obj *readObjPtr (in/out)</td>
<td>A pointer to a Tcl Object in which to store the characters read from the channel.</td>
</tr>
<tr>
<td>int charsToRead (in)</td>
<td>The number of characters to read from the channel. If the channel's encoding is binary, this is equivalent to the number of bytes to read from the channel.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>int <strong>appendFlag</strong> (in)</td>
<td>If non-zero, data read from the channel will be appended to the object.</td>
</tr>
<tr>
<td></td>
<td>Otherwise, the data will replace the existing contents of the object.</td>
</tr>
<tr>
<td>char <strong>readBuf</strong> (out)</td>
<td>A buffer in which to store the bytes read from the channel.</td>
</tr>
<tr>
<td>int <strong>bytesToRead</strong> (in)</td>
<td>The number of bytes to read from the channel.</td>
</tr>
<tr>
<td></td>
<td>The buffer <strong>readBuf</strong> must be large enough to hold this many bytes.</td>
</tr>
<tr>
<td>Tcl_Obj <strong>lineObjPtr</strong> (in/out)</td>
<td>A pointer to a Tcl object in which to store the line read from the channel.</td>
</tr>
<tr>
<td></td>
<td>The line read will be appended to the current value of the object.</td>
</tr>
<tr>
<td>Tcl_DString <strong>lineRead</strong> (in/out)</td>
<td>A pointer to a Tcl dynamic string in which to store the line read from the channel.</td>
</tr>
<tr>
<td></td>
<td>Must have been initialized by the caller.</td>
</tr>
<tr>
<td>const char <strong>input</strong> (in)</td>
<td>The input to add to a channel buffer.</td>
</tr>
<tr>
<td>int <strong>inputLen</strong> (in)</td>
<td>Length of the input</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>int <strong>addAtEnd</strong> (in)</td>
<td>Flag indicating whether the input should be added to the end or beginning of the channel buffer.</td>
</tr>
<tr>
<td>Tcl_Obj <strong>writeObjPtr</strong> (in)</td>
<td>A pointer to a Tcl Object whose contents will be output to the channel.</td>
</tr>
<tr>
<td>const char <strong>charBuf</strong> (in)</td>
<td>A buffer containing the characters to output to the channel.</td>
</tr>
<tr>
<td>const char <strong>byteBuf</strong> (in)</td>
<td>A buffer containing the bytes to output to the channel.</td>
</tr>
<tr>
<td>int <strong>bytesToWrite</strong> (in)</td>
<td>The number of bytes to consume from <strong>charBuf</strong> or <strong>byteBuf</strong> and output to the channel.</td>
</tr>
<tr>
<td>Tcl_WideInt <strong>offset</strong> (in)</td>
<td>How far to move the access point in the channel at which the next input or output operation will be applied, measured in bytes from the position given by <strong>seekMode</strong>. May be either positive or negative.</td>
</tr>
<tr>
<td>int <strong>seekMode</strong> (in)</td>
<td>Relative to which point to seek; used with <strong>offset</strong> to calculate the new access point for the channel. Legal values are</td>
</tr>
</tbody>
</table>
SEEK_SET, SEEK_CUR, and SEEK_END.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_WideInt length</strong></td>
<td>(in) The (non-negative) length to truncate the channel the channel to.</td>
</tr>
<tr>
<td>*<em>const char <em>optionName</em></em></td>
<td>(in) The name of an option applicable to this channel, such as -blocking. May have any of the values accepted by the fconfigure command.</td>
</tr>
<tr>
<td>*<em>Tcl_DString <em>optionValue</em></em></td>
<td>(in) Where to store the value of an option or a list of all options and their values. Must have been initialized by the caller.</td>
</tr>
<tr>
<td>*<em>const char <em>newValue</em></em></td>
<td>(in) New value for the option given by optionName.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

The Tcl channel mechanism provides a device-independent and platform-independent mechanism for performing buffered input and output operations on a variety of file, socket, and device types. The channel mechanism is extensible to new channel types, by providing a low-level channel driver for the new type; the channel driver interface is described in the manual entry for *Tcl_CreateChannel*. The channel mechanism provides a buffering scheme modeled after Unix's standard I/O, and it also allows for nonblocking I/O on channels.

The procedures described in this manual entry comprise the C APIs of the generic layer of the channel architecture. For a description of the
channel driver architecture and how to implement channel drivers for new types of channels, see the manual entry for Tcl_CreateChannel.

### TCL_OPENFILECHANNEL

**Tcl_OpenFileChannel** opens a file specified by fileName and returns a channel handle that can be used to perform input and output on the file. This API is modeled after the fopen procedure of the Unix standard I/O library. The syntax and meaning of all arguments is similar to those given in the Tcl open command when opening a file. If an error occurs while opening the channel, **Tcl_OpenFileChannel** returns NULL and records a POSIX error code that can be retrieved with **Tcl_GetErrno**. In addition, if interp is non-NULL, **Tcl_OpenFileChannel** leaves an error message in interp's result after any error. As of Tcl 8.4, the object-based API **Tcl_FSOpenFileChannel** should be used in preference to **Tcl_OpenFileChannel** wherever possible.

The newly created channel is not registered in the supplied interpreter; to register it, use **Tcl_RegisterChannel**, described below. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

### TCL_OPENCOMMANDCHANNEL

**Tcl_OpenCommandChannel** provides a C-level interface to the functions of the exec and open commands. It creates a sequence of subprocesses specified by the argv and argc arguments and returns a channel that can be used to communicate with these subprocesses. The flags argument indicates what sort of communication will exist with the command pipeline.

If the TCL_STDIN flag is set then the standard input for the first subprocess will be tied to the channel: writing to the channel will provide input to the subprocess. If TCL_STDIN is not set, then standard input for the first subprocess will be the same as this application's standard input. If TCL_STDOUT is set then standard output from the last subprocess can be read from the channel; otherwise it goes to this
application's standard output. If TCL_STDERR is set, standard error output for all subprocesses is returned to the channel and results in an error when the channel is closed; otherwise it goes to this application's standard error. If TCL_ENFORCE_MODE is not set, then argc and argv can redirect the stdio handles to override TCL_STDIN, TCL_STDOUT, and TCL_STDERR; if it is set, then it is an error for argc and argv to override stdio channels for which TCL_STDIN, TCL_STDOUT, and TCL_STDERR have been set.

If an error occurs while opening the channel, Tcl_OpenCommandChannel returns NULL and records a POSIX error code that can be retrieved with Tcl_GetErrno. In addition, Tcl_OpenCommandChannel leaves an error message in the interpreter's result if interp is not NULL.

The newly created channel is not registered in the supplied interpreter; to register it, use Tcl_RegisterChannel, described below. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

**TCL_MAKEFILECHANNEL**

Tcl_MakeFileChannel makes a Tcl_Channel from an existing, platform-specific, file handle. The newly created channel is not registered in the supplied interpreter; to register it, use Tcl_RegisterChannel, described below. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

**TCL_GETCHANNEL**

Tcl_GetChannel returns a channel given the channelName used to create it with Tcl_CreateChannel and a pointer to a Tcl interpreter in interp. If a channel by that name is not registered in that interpreter, the procedure returns NULL. If the modePtr argument is not NULL, it points at an integer variable that will receive an OR-ed combination of
**TCL_READABLE** and **TCL_WRITABLE** describing whether the channel is open for reading and writing.

**Tcl_GetChannelNames** and **Tcl_GetChannelNamesEx** write the names of the registered channels to the interpreter's result as a list object. **Tcl_GetChannelNamesEx** will filter these names according to the **pattern**. If **pattern** is NULL, then it will not do any filtering. The return value is **TCL_OK** if no errors occurred writing to the result, otherwise it is **TCL_ERROR**, and the error message is left in the interpreter's result.

**TCL_REGISTERCHANNEL**

**Tcl_RegisterChannel** adds a channel to the set of channels accessible in **interp**. After this call, Tcl programs executing in that interpreter can refer to the channel in input or output operations using the name given in the call to **Tcl_CreateChannel**. After this call, the channel becomes the property of the interpreter, and the caller should not call **Tcl_Close** for the channel; the channel will be closed automatically when it is unregistered from the interpreter.

Code executing outside of any Tcl interpreter can call **Tcl_RegisterChannel** with **interp** as NULL, to indicate that it wishes to hold a reference to this channel. Subsequently, the channel can be registered in a Tcl interpreter and it will only be closed when the matching number of calls to **Tcl_UnregisterChannel** have been made. This allows code executing outside of any interpreter to safely hold a reference to a channel that is also registered in a Tcl interpreter.

This procedure interacts with the code managing the standard channels. If no standard channels were initialized before the first call to **Tcl_RegisterChannel**, they will get initialized by that call. See **Tcl_StandardChannels** for a general treatise about standard channels and the behaviour of the Tcl library with regard to them.

**TCL_UNREGISTERCHANNEL**

**Tcl_UnregisterChannel** removes a channel from the set of channels accessible in **interp**. After this call, Tcl programs will no longer be able to
use the channel's name to refer to the channel in that interpreter. If this operation removed the last registration of the channel in any interpreter, the channel is also closed and destroyed.

Code not associated with a Tcl interpreter can call **Tcl_UnregisterChannel** with interp as NULL, to indicate to Tcl that it no longer holds a reference to that channel. If this is the last reference to the channel, it will now be closed. **Tcl_UnregisterChannel** is very similar to **Tcl_DetachChannel** except that it will also close the channel if no further references to it exist.

### TCL_DETACHCHANNEL

**Tcl_DetachChannel** removes a channel from the set of channels accessible in interp. After this call, Tcl programs will no longer be able to use the channel's name to refer to the channel in that interpreter. Beyond that, this command has no further effect. It cannot be used on the standard channels (stdout, stderr, stdin), and will return **TCL_ERROR** if passed one of those channels.

Code not associated with a Tcl interpreter can call **Tcl_DetachChannel** with interp as NULL, to indicate to Tcl that it no longer holds a reference to that channel. If this is the last reference to the channel, unlike **Tcl_UnregisterChannel**, it will not be closed.

### TCL_ISSTANDARDCHANNEL

**Tcl_IsStandardChannel** tests whether a channel is one of the three standard channels, stdin, stdout or stderr. If so, it returns 1, otherwise 0.

No attempt is made to check whether the given channel or the standard channels are initialized or otherwise valid.

### TCL_CLOSE

**Tcl_Close** destroys the channel channel, which must denote a currently open channel. The channel should not be registered in any interpreter when **Tcl_Close** is called. Buffered output is flushed to the channel's
output device prior to destroying the channel, and any buffered input is discarded. If this is a blocking channel, the call does not return until all buffered data is successfully sent to the channel's output device. If this is a nonblocking channel and there is buffered output that cannot be written without blocking, the call returns immediately; output is flushed in the background and the channel will be closed once all of the buffered data has been output. In this case errors during flushing are not reported.

If the channel was closed successfully, \texttt{Tcl\_Close} returns \texttt{TCL\_OK}. If an error occurs, \texttt{Tcl\_Close} returns \texttt{TCL\_ERROR} and records a POSIX error code that can be retrieved with \texttt{Tcl\_GetErrno}. If the channel is being closed synchronously and an error occurs during closing of the channel and interp is not NULL, an error message is left in the interpreter's result.

Note: it is not safe to call \texttt{Tcl\_Close} on a channel that has been registered using \texttt{Tcl\_RegisterChannel}; see the documentation for \texttt{Tcl\_RegisterChannel}, above, for details. If the channel has ever been given as the \texttt{chan} argument in a call to \texttt{Tcl\_RegisterChannel}, you should instead use \texttt{Tcl\_UnregisterChannel}, which will internally call \texttt{Tcl\_Close} when all calls to \texttt{Tcl\_RegisterChannel} have been matched by corresponding calls to \texttt{Tcl\_UnregisterChannel}.

\section*{TCL\_READCHARS AND TCL\_READ}

\texttt{Tcl\_ReadChars} consumes bytes from \textit{channel}, converting the bytes to UTF-8 based on the channel's encoding and storing the produced data in \textit{readObjPtr}'s string representation. The return value of \texttt{Tcl\_ReadChars} is the number of characters, up to \textit{charsToRead}, that were stored in \textit{readObjPtr}. If an error occurs while reading, the return value is -1 and \texttt{Tcl\_ReadChars} records a POSIX error code that can be retrieved with \texttt{Tcl\_GetErrno}.

Setting \textit{charsToRead} to -1 will cause the command to read all characters currently available (non-blocking) or everything until eof (blocking mode).
The return value may be smaller than the value to read, indicating that less data than requested was available. This is called a short read. In blocking mode, this can only happen on an end-of-file. In nonblocking mode, a short read can also occur if there is not enough input currently available: **Tcl_ReadChars** returns a short count rather than waiting for more data.

If the channel is in blocking mode, a return value of zero indicates an end-of-file condition. If the channel is in nonblocking mode, a return value of zero indicates either that no input is currently available or an end-of-file condition. Use **Tcl_Eof** and **Tcl_InputBlocked** to tell which of these conditions actually occurred.

**Tcl_ReadChars** translates the various end-of-line representations into the canonical in internal representation according to the current end-of-line recognition mode. End-of-line recognition and the various platform-specific modes are described in the manual entry for the Tcl **fconfigure** command.

As a performance optimization, when reading from a channel with the encoding binary, the bytes are not converted to UTF-8 as they are read. Instead, they are stored in **readObjPtr**'s internal representation as a byte-array object. The string representation of this object will only be constructed if it is needed (e.g., because of a call to **Tcl_GetStringFromObj**). In this way, byte-oriented data can be read from a channel, manipulated by calling **Tcl_GetByteArrayFromObj** and related functions, and then written to a channel without the expense of ever converting to or from UTF-8.

**Tcl_Read** is similar to **Tcl_ReadChars**, except that it does not do encoding conversions, regardless of the channel's encoding. It is deprecated and exists for backwards compatibility with non-internationalized Tcl extensions. It consumes bytes from channel and stores them in readBuf, performing end-of-line translations on the way. The return value of **Tcl_Read** is the number of bytes, up to bytesToRead, written in readBuf. The buffer produced by **Tcl_Read** is not null-terminated. Its contents are valid from the zeroth position up to and excluding the position indicated by the return value.
**Tcl_ReadRaw** is the same as **Tcl_Read** but does not compensate for stacking. While **Tcl_Read** (and the other functions in the API) always get their data from the topmost channel in the stack the supplied channel is part of, **Tcl_ReadRaw** does not. Thus this function is only usable for transformational channel drivers, i.e. drivers used in the middle of a stack of channels, to move data from the channel below into the transformation.

---

**TCL_GETSOBJ AND TCL_GETS**

**Tcl_GetsObj** consumes bytes from *channel*, converting the bytes to UTF-8 based on the channel's encoding, until a full line of input has been seen. If the channel's encoding is *binary*, each byte read from the channel is treated as an individual Unicode character. All of the characters of the line except for the terminating end-of-line character(s) are appended to *lineObjPtr's* string representation. The end-of-line character(s) are read and discarded.

If a line was successfully read, the return value is greater than or equal to zero and indicates the number of bytes stored in *lineObjPtr*. If an error occurs, **Tcl_GetsObj** returns -1 and records a POSIX error code that can be retrieved with **Tcl_GetErrno**. **Tcl_GetsObj** also returns -1 if the end of the file is reached; the **Tcl_Eof** procedure can be used to distinguish an error from an end-of-file condition.

If the channel is in nonblocking mode, the return value can also be -1 if no data was available or the data that was available did not contain an end-of-line character. When -1 is returned, the **Tcl_InputBlocked** procedure may be invoked to determine if the channel is blocked because of input unavailability.

**Tcl_Gets** is the same as **Tcl_GetsObj** except the resulting characters are appended to the dynamic string given by *lineRead* rather than a Tcl object.

---

**TCL_UNGETS**

**Tcl_Ungets** is used to add data to the input queue of a channel, at
either the head or tail of the queue. The pointer input points to the data that is to be added. The length of the input to add is given by inputLen. A non-zero value of addAtEnd indicates that the data is to be added at the end of queue; otherwise it will be added at the head of the queue. If channel has a “sticky” EOF set, no data will be added to the input queue. Tcl_Ungets returns inputLen or -1 if an error occurs.

**TCL_WRITECHARS, TCL_WRITEOBJ, AND TCL_WRITE**

**Tcl_WriteChars** accepts bytesToWrite bytes of character data at charBuf. The UTF-8 characters in the buffer are converted to the channel's encoding and queued for output to channel. If bytesToWrite is negative, Tcl_WriteChars expects charBuf to be null-terminated and it outputs everything up to the null.

Data queued for output may not appear on the output device immediately, due to internal buffering. If the data should appear immediately, call **Tcl_Flush** after the call to Tcl_WriteChars, or set the -buffering option on the channel to none. If you wish the data to appear as soon as a complete line is accepted for output, set the -buffering option on the channel to line mode.

The return value of **Tcl_WriteChars** is a count of how many bytes were accepted for output to the channel. This is either greater than zero to indicate success or -1 to indicate that an error occurred. If an error occurs, **Tcl_WriteChars** records a POSIX error code that may be retrieved with **Tcl_GetErrno**.

Newline characters in the output data are translated to platform-specific end-of-line sequences according to the -translation option for the channel. This is done even if the channel has no encoding.

**Tcl_WriteObj** is similar to **Tcl_WriteChars** except it accepts a Tcl object whose contents will be output to the channel. The UTF-8 characters in writeObjPtr's string representation are converted to the channel's encoding and queued for output to channel. As a performance optimization, when writing to a channel with the encoding binary, UTF-8 characters are not converted as they are written. Instead, the bytes in
writeObjPtr's internal representation as a byte-array object are written to the channel. The byte-array representation of the object will be constructed if it is needed. In this way, byte-oriented data can be read from a channel, manipulated by calling Tcl_GetByteArrayFromObj and related functions, and then written to a channel without the expense of ever converting to or from UTF-8.

Tcl_Write is similar to Tcl_WriteChars except that it does not do encoding conversions, regardless of the channel's encoding. It is deprecated and exists for backwards compatibility with non-internationalized Tcl extensions. It accepts bytesToWrite bytes of data at byteBuf and queues them for output to channel. If bytesToWrite is negative, Tcl_Write expects byteBuf to be null-terminated and it outputs everything up to the null.

Tcl_WriteRaw is the same as Tcl_Write but does not compensate for stacking. While Tcl_Write (and the other functions in the API) always feed their input to the topmost channel in the stack the supplied channel is part of, Tcl_WriteRaw does not. Thus this function is only usable for transformational channel drivers, i.e. drivers used in the middle of a stack of channels, to move data from the transformation into the channel below it.

TCL_FLUSH

Tcl_Flush causes all of the buffered output data for channel to be written to its underlying file or device as soon as possible. If the channel is in blocking mode, the call does not return until all the buffered data has been sent to the channel or some error occurred. The call returns immediately if the channel is nonblocking; it starts a background flush that will write the buffered data to the channel eventually, as fast as the channel is able to absorb it.

The return value is normally TCL_OK. If an error occurs, Tcl_Flush returns TCL_ERROR and records a POSIX error code that can be retrieved with Tcl_GetErrno.

TCL_SEEK
**Tcl.Seek** moves the access point in *channel* where subsequent data will be read or written. Buffered output is flushed to the channel and buffered input is discarded, prior to the seek operation.

*Tcl.Seek* normally returns the new access point. If an error occurs, *Tcl.Seek* returns -1 and records a POSIX error code that can be retrieved with *Tcl.GetErrno*. After an error, the access point may or may not have been moved.

---

**TCL_TELL**

*Tcl.Tell* returns the current access point for a channel. The returned value is -1 if the channel does not support seeking.

---

**TCL_TRUNCATECHANNEL**

*Tcl.TruncateChannel* truncates the file underlying *channel* to a given length of bytes. It returns *TCL_OK* if the operation succeeded, and *TCL_ERROR* otherwise.

---

**TCL_GETCHANNELOPTION**

*Tcl.GetChannelOption* retrieves, in *optionValue*, the value of one of the options currently in effect for a channel, or a list of all options and their values. The *channel* argument identifies the channel for which to query an option or retrieve all options and their values. If *optionName* is not NULL, it is the name of the option to query; the option's value is copied to the Tcl dynamic string denoted by *optionValue*. If *optionName* is NULL, the function stores an alternating list of option names and their values in *optionValue*, using a series of calls to *Tcl_DStringAppendElement*. The various preexisting options and their possible values are described in the manual entry for the Tcl *fconfigure* command. Other options can be added by each channel type. These channel type specific options are described in the manual entry for the Tcl command that creates a channel of that type; for example, the additional options for TCP based channels are described in the manual entry for the Tcl *socket* command. The procedure normally returns *TCL_OK*. If an error occurs, it returns *TCL_ERROR* and calls
**Tcl_SetErrno** to store an appropriate POSIX error code.

**TCL_SETCHANNELOPTION**

*Tcl_SetChannelOption* sets a new value *newValue* for an option *optionName* on *channel*. The procedure normally returns **TCL_OK**. If an error occurs, it returns **TCL_ERROR**; in addition, if *interp* is non-NULL, *Tcl_SetChannelOption* leaves an error message in the interpreter's result.

**TCL_EOF**

*Tcl_Eof* returns a nonzero value if *channel* encountered an end of file during the last input operation.

**TCL_INPUTBLOCKED**

*Tcl_InputBlocked* returns a nonzero value if *channel* is in nonblocking mode and the last input operation returned less data than requested because there was insufficient data available. The call always returns zero if the channel is in blocking mode.

**TCL_INPUTBUFFERED**

*Tcl_InputBuffered* returns the number of bytes of input currently buffered in the internal buffers for a channel. If the channel is not open for reading, this function always returns zero.

**TCL_OUTPUTBUFFERED**

*Tcl_OutputBuffered* returns the number of bytes of output currently buffered in the internal buffers for a channel. If the channel is not open for writing, this function always returns zero.

**PLATFORM ISSUES**

The handles returned from *Tcl_GetChannelHandle* depend on the platform and the channel type. On Unix platforms, the handle is always a Unix file descriptor as returned from the *open* system call. On
Windows platforms, the handle is a file HANDLE when the channel was created with Tcl_OpenFileChannel, Tcl_OpenCommandChannel, or Tcl_MakeFileChannel. Other channel types may return a different type of handle on Windows platforms.

**SEE ALSO**

DString, fconfigure, filename, fopen, Tcl_CreateChannel

**KEYWORDS**

access point, blocking, buffered I/O, channel, channel driver, end of file, flush, input, nonblocking, output, read, seek, write

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996-1997 Sun Microsystems, Inc.
NAME
Tcl_Access, Tcl_Stat - check file permissions and other attributes

SYNOPSIS
#include <tcl.h>
int
Tcl_Access(path, mode)
int
Tcl_Stat(path, statPtr)

ARGUMENTS
DESCRIPTION
KEYWORDS

Native name of the file to check the attributes of.

Mask consisting of one or more of R_OK, W_OK,
X_OK and F_OK. R_OK, W_OK and X_OK request checking whether the file exists and has read, write and execute permissions, respectively. F_OK just requests checking for the existence of the file.

| struct stat *statPtr (out) | The structure that contains the result. |

**DESCRIPTION**

As of Tcl 8.4, the object-based APIs `Tcl_FSAccess` and `Tcl_FSStat` should be used in preference to `Tcl_Access` and `Tcl_Stat`, wherever possible.

There are two reasons for calling `Tcl_Access` and `Tcl_Stat` rather than calling system level functions `access` and `stat` directly. First, the Windows implementation of both functions fixes some bugs in the system level calls. Second, both `Tcl_Access` and `Tcl_Stat` (as well as `Tcl_OpenFileChannelProc`) hook into a linked list of functions. This allows the possibility to reroute file access to alternative media or access methods.

`Tcl_Access` checks whether the process would be allowed to read, write or test for existence of the file (or other file system object) whose name is pathname. If pathname is a symbolic link on Unix, then permissions of the file referred by this symbolic link are tested.

On success (all requested permissions granted), zero is returned. On error (at least one bit in mode asked for a permission that is denied, or some other error occurred), -1 is returned.

`Tcl_Stat` fills the stat structure `statPtr` with information about the specified file. You do not need any access rights to the file to get this
information but you need search rights to all directories named in the path leading to the file. The stat structure includes info regarding device, inode (always 0 on Windows), privilege mode, nlink (always 1 on Windows), user id (always 0 on Windows), group id (always 0 on Windows), rdev (same as device on Windows), size, last access time, last modification time, and creation time.

If \emph{path} exists, \texttt{Tcl Stat} returns 0 and the stat structure is filled with data. Otherwise, -1 is returned, and no stat info is given.

**KEYWORDS**

\texttt{stat, access}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1998-1999 Scriptics Corporation
NAME
Tcl_GetReturnOptions, Tcl_SetReturnOptions,
Tcl_AddErrorInfo, Tcl_AppendObjToErrorInfo,
Tcl_AddObjErrorInfo, Tcl_SetObjErrorCode, Tcl_SetErrorCode,
Tcl_SetErrorCodeVA, Tcl_PosixError, Tcl_LogCommandInfo -
retrieve or record information about errors and other return
options
SYNOPSIS
#include <tcl.h>
Tcl_Obj *
Tcl_GetReturnOptions(interp, code)
int
Tcl_SetReturnOptions(interp, options)
Tcl_AddErrorInfo(interp, message)
Tcl_AppendObjToErrorInfo(interp, objPtr)
Tcl_AddObjErrorInfo(interp, message, length)
Tcl_SetObjErrorCode(interp, errorObjPtr)
Tcl_SetErrorCode(interp, element, element, ... (char *) NULL)
Tcl_SetErrorCodeVA(interp, argList)
const char *
Tcl_PosixError(interp)
void
Tcl_LogCommandInfo(interp, script, command, commandLength)
ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS
NAME
SYNOPSIS

#include <tcl.h>

Tcl_Obj *Tcl_GetReturnOptions(interp, code)
int Tcl_SetReturnOptions(interp, options)
Tcl_AddErrorInfo(interp, message)
Tcl_AppendObjToErrorInfo(interp, objPtr)
Tcl_AddObjErrorInfo(interp, message, length)
Tcl_SetObjErrorCode(interp, errorObjPtr)
Tcl_SetErrorCode(interp, element, ... (char *) NULL)
Tcl_SetErrorCodeVA(interp, argList)
const char *Tcl_PosixError(interp)
void Tcl_LogCommandInfo(interp, script, command, commandLength)

ARGUMENTS

Tcl_Interp *interp (in)     Interpreter in which to record information.
int code ()         The code returned from script evaluation.
Tcl_Obj *options ()   A dictionary of return options.
char *message (in)   For Tcl_AddErrorInfo, this is a conventional C
string to append to the -errorinfo return option. For **Tcl_AddObjErrorInfo**, this points to the first byte of an array of `length` bytes containing a string to append to the -errorinfo return option. This byte array may contain embedded null bytes unless `length` is negative.

<table>
<thead>
<tr>
<th>**Tcl_Obj **objPtr (in)</th>
<th>A message to be appended to the -errorinfo return option in the form of a Tcl_Obj value.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>int length (in)</strong></td>
<td>The number of bytes to copy from <code>message</code> when appending to the -errorinfo return option. If negative, all bytes up to the first null byte are used.</td>
</tr>
<tr>
<td>**Tcl_Obj **errorObjPtr (in)</td>
<td>The -errorcode return option will be set to this value.</td>
</tr>
<tr>
<td>**char **element (in)</td>
<td>String to record as one element of the -errorcode return option. Last <code>element</code> argument must be NULL.</td>
</tr>
<tr>
<td><strong>va_list argList (in)</strong></td>
<td>An argument list which must have been initialized using <strong>va_start</strong>, and</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>const char *script (in)</td>
<td>Pointer to first character in script containing command (must be &lt;= command)</td>
</tr>
<tr>
<td>const char *command (in)</td>
<td>Pointer to first character in command that generated the error</td>
</tr>
<tr>
<td>int commandLength (in)</td>
<td>Number of bytes in command; -1 means use all bytes up to first null byte</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

The `Tcl_SetReturnOptions` and `Tcl_GetReturnOptions` routines expose the same capabilities as the `return` and `catch` commands, respectively, in the form of a C interface.

`Tcl_GetReturnOptions` retrieves the dictionary of return options from an interpreter following a script evaluation. Routines such as `Tcl_Eval` are called to evaluate a script in an interpreter. These routines return an integer completion code. These routines also leave in the interpreter both a result and a dictionary of return options generated by script evaluation. Just as `Tcl_GetObjResult` retrieves the result, `Tcl_GetReturnOptions` retrieves the dictionary of return options. The integer completion code should be passed as the `code` argument to `Tcl_GetReturnOptions` so that all required options will be present in the dictionary. Specifically, a `code` value of `TCL_ERROR` will ensure that entries for the keys `-errorinfo`, `-errorcode`, and `-errorline` will appear in the dictionary. Also, the entries for the keys `-code` and `-level` will be adjusted if necessary to agree with the value of `code`. The `(Tcl_Obj *)` returned by `Tcl_GetReturnOptions` points to an unshared `Tcl_Obj` with reference count of zero. The dictionary may be written to,
either adding, removing, or overwriting any entries in it, with the need to check for a shared object.

A typical usage for `Tcl_GetReturnOptions` is to retrieve the stack trace when script evaluation returns `TCL_ERROR`, like so:

```c
int code = Tcl_Eval(interp, script);
if (code == TCL_ERROR) {
    Tcl_Obj *options = Tcl_GetReturnOptions(interp, code);
    Tcl_Obj *key = Tcl_NewStringObj("-errorinfo", -1);
    Tcl_Obj *stackTrace;
    Tcl_IncrRefCount(key);
    Tcl_DictObjGet(NULL, options, key, &stackTrace);
    Tcl_DecrRefCount(key);
    /* Do something with stackTrace */
}
```

`Tcl_SetReturnOptions` sets the return options of `interp` to be `options`. If `options` contains any invalid value for any key, `TCL_ERROR` will be returned, and the interp result will be set to an appropriate error message. Otherwise, a completion code in agreement with the `-code` and `-level` keys in `options` will be returned.

As an example, Tcl's `return` command itself could be implemented in terms of `Tcl_SetReturnOptions` like so:

```c
if ((objc % 2) == 0) { /* explicit result argument */
    objc--;
    Tcl_SetObjResult(interp, objv[objc]);
}
return Tcl_SetReturnOptions(interp, Tcl_NewListObj(objc-1, objv+1));
```

(It is not really implemented that way. Internal access privileges allow
for a more efficient alternative that meshes better with the bytecode compiler.)

Note that a newly created Tcl_Obj may be passed in as the options argument without the need to tend to any reference counting. This is analogous to Tcl_SetObjResult.

While Tcl_SetReturnOptions provides a general interface to set any collection of return options, there are a handful of return options that are very frequently used. Most notably the -errorinfo and -errorcode return options should be set properly when the command procedure of a command returns TCL_ERROR. Tcl provides several simpler interfaces to more directly set these return options.

The -errorinfo option holds a stack trace of the operations that were in progress when an error occurred, and is intended to be human-readable. The -errorcode option holds a list of items that are intended to be machine-readable. The first item in the -errorcode value identifies the class of error that occurred (e.g. POSIX means an error occurred in a POSIX system call) and additional elements hold additional pieces of information that depend on the class. See the tclvars manual entry for details on the various formats for the -errorcode option used by Tcl's built-in commands.

The -errorinfo option value is gradually built up as an error unwinds through the nested operations. Each time an error code is returned to Tcl_Eval, or any of the routines that performs script evaluation, the procedure Tcl_AddErrorInfo is called to add additional text to the -errorinfo value describing the command that was being executed when the error occurred. By the time the error has been passed all the way back to the application, it will contain a complete trace of the activity in progress when the error occurred.

It is sometimes useful to add additional information to the -errorinfo value beyond what can be supplied automatically by the script evaluation routines. Tcl_AddErrorInfo may be used for this purpose: its message argument is an additional string to be appended to the -errorinfo option. For example, when an error arises during the source
command, the procedure **Tcl_AddErrorInfo** is called to record the name of the file being processed and the line number on which the error occurred. Likewise, when an error arises during evaluation of a Tcl procedures, the procedure name and line number within the procedure are recorded, and so on. The best time to call **Tcl_AddErrorInfo** is just after a script evaluation routine has returned **TCL_ERROR**. The value of the **-errorline** return option (retrieved via a call to **Tcl_GetReturnOptions**) often makes up a useful part of the message passed to **Tcl_AddErrorInfo**.

**Tcl_AppendObjToErrorInfo** is an alternative interface to the same functionality as **Tcl_AddErrorInfo**. **Tcl_AppendObjToErrorInfo** is called when the string value to be appended to the **-errorinfo** option is available as a **Tcl_Obj** instead of as a **char** array.

**Tcl_AddObjErrorInfo** is nearly identical to **Tcl_AddErrorInfo**, except that it has an additional **length** argument. This allows the message string to contain embedded null bytes. This is essentially never a good idea. If the message needs to contain the null character **U+0000**, Tcl's usual internal encoding rules should be used to avoid the need for a null byte. If the **Tcl_AddObjErrorInfo** interface is used at all, it should be with a negative **length** value.

The procedure **Tcl_SetObjErrorCode** is used to set the **-errorcode** return option to the list object **errorObjPtr** built up by the caller. **Tcl_SetObjErrorCode** is typically invoked just before returning an error. If an error is returned without calling **Tcl_SetObjErrorCode** or **Tcl_SetErrorCode** the Tcl interpreter automatically sets the **-errorcode** return option to **NONE**.

The procedure **Tcl_SetErrorCode** is also used to set the **-errorcode** return option. However, it takes one or more strings to record instead of an object. Otherwise, it is similar to **Tcl_SetObjErrorCode** in behavior.

**Tcl_SetErrorCodeVA** is the same as **Tcl_SetErrorCode** except that instead of taking a variable number of arguments it takes an argument list.
Tcl_PosixError sets the -errorcode variable after an error in a POSIX kernel call. It reads the value of the errno C variable and calls Tcl_SetErrorCode to set the -errorcode return option in the POSIX format. The caller must previously have called Tcl_SetErrno to set errno; this is necessary on some platforms (e.g. Windows) where Tcl is linked into an application as a shared library, or when the error occurs in a dynamically loaded extension. See the manual entry for Tcl_SetErrno for more information.

Tcl_PosixError returns a human-readable diagnostic message for the error (this is the same value that will appear as the third element in the -errorcode value). It may be convenient to include this string as part of the error message returned to the application in the interpreter's result.

Tcl_LogCommandInfo is invoked after an error occurs in an interpreter. It adds information about the command that was being executed when the error occurred to the -errorinfo value, and the line number stored internally in the interpreter is set.

In older releases of Tcl, there was no Tcl_GetReturnOptions routine. In its place, the global Tcl variables errorInfo and errorCode were the only place to retrieve the error information. Much existing code written for older Tcl releases still access this information via those global variables.

It is important to realize that while reading from those global variables remains a supported way to access these return option values, it is important not to assume that writing to those global variables will properly set the corresponding return options. It has long been emphasized in this manual page that it is important to call the procedures described here rather than setting errorInfo or errorCode directly with Tcl_ObjSetVar2.

If the procedure Tcl_ResetResult is called, it clears all of the state of the interpreter associated with script evaluation, including the entire return options dictionary. In particular, the -errorinfo and -errorcode options are reset. If an error had occurred, the Tcl_ResetResult call will clear the error state to make it appear as if no error had occurred after
all. The global variables `errorInfo` and `errorCode` are not modified by `Tcl_ResetResult` so they continue to hold a record of information about the most recent error seen in an interpreter.

**SEE ALSO**

`Tcl_DecrRefCount`, `Tcl_IncrRefCount`, `Tcl_Interp`, `Tcl_ResetResult`, `Tcl_SetErrno`

**KEYWORDS**

`error`, `object`, `object result`, `stack`, `trace`, `variable`
NAME
Tcl_OpenTcpClient, Tcl_MakeTcpClientChannel, Tcl_OpenTcpServer - procedures to open channels using TCP sockets

SYNOPSIS
#include <tcl.h>
Tcl_Channel
Tcl_OpenTcpClient(interp, port, host, myaddr, myport, async)
Tcl_Channel
Tcl_MakeTcpClientChannel(sock)
Tcl_Channel
Tcl_OpenTcpServer(interp, port, myaddr, proc, clientData)

ARGUMENTS

DESCRIPTION

PLATFORM ISSUES

SEE ALSO

KEYWORDS

NAME
Tcl_OpenTcpClient, Tcl_MakeTcpClientChannel, Tcl_OpenTcpServer - procedures to open channels using TCP sockets

SYNOPSIS
#include <tcl.h>
Tcl_Channel
Tcl_OpenTcpClient(interp, port, host, myaddr, myport, async)
Tcl_Channel
**Tcl\_MakeTcpClientChannel**(*sock*)

**Tcl\_Channel**

**Tcl\_OpenTcpServer**(*interp, port, myaddr, proc, clientData*)

### ARGUMENTS

<table>
<thead>
<tr>
<th><strong>Tcl_Interp</strong> <em>interp</em> (in)</th>
<th>Tcl interpreter to use for error reporting. If non-NULL and an error occurs, an error message is left in the interpreter's result.</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <em>port</em> (in)</td>
<td>A port number to connect to as a client or to listen on as a server.</td>
</tr>
<tr>
<td>const char <em>host</em> (in)</td>
<td>A string specifying a host name or address for the remote end of the connection.</td>
</tr>
<tr>
<td>int <em>myport</em> (in)</td>
<td>A port number for the client's end of the socket. If 0, a port number is allocated at random.</td>
</tr>
<tr>
<td>const char <em>myaddr</em> (in)</td>
<td>A string specifying the host name or address for network interface to use for the local end of the connection. If NULL, a default interface is chosen.</td>
</tr>
<tr>
<td>int <em>async</em> (in)</td>
<td>If nonzero, the client socket is connected asynchronously to the server.</td>
</tr>
</tbody>
</table>
ClientData **sock** (in) | Platform-specific handle for client TCP socket.
---|---
*Tcl_TcpAcceptProc* **proc** (in) | Pointer to a procedure to invoke each time a new connection is accepted via the socket.
ClientData **clientData** (in) | Arbitrary one-word value to pass to *proc*.

**DESCRIPTION**

These functions are convenience procedures for creating channels that communicate over TCP sockets. The operations on a channel are described in the manual entry for *Tcl_OpenFileChannel*.

**TCL_OPENTCPCLIENT**

*Tcl_OpenTcpClient* opens a client TCP socket connected to a *port* on a specific *host*, and returns a channel that can be used to communicate with the server. The host to connect to can be specified either as a domain name style name (e.g. **www.sunlabs.com**), or as a string containing the alphanumeric representation of its four-byte address (e.g. **127.0.0.1**). Use the string **localhost** to connect to a TCP socket on the host on which the function is invoked.

The *myaddr* and *myport* arguments allow a client to specify an address for the local end of the connection. If *myaddr* is NULL, then an interface is chosen automatically by the operating system. If *myport* is 0, then a port number is chosen at random by the operating system.

If *async* is zero, the call to *Tcl_OpenTcpClient* returns only after the client socket has either successfully connected to the server, or the attempted connection has failed. If *async* is nonzero the socket is connected asynchronously and the returned channel may not yet be connected to the server when the call to *Tcl_OpenTcpClient* returns. If
the channel is in blocking mode and an input or output operation is done on the channel before the connection is completed or fails, that operation will wait until the connection either completes successfully or fails. If the channel is in nonblocking mode, the input or output operation will return immediately and a subsequent call to \texttt{Tcl\_InputBlocked} on the channel will return nonzero.

The returned channel is opened for reading and writing. If an error occurs in opening the socket, \texttt{Tcl\_OpenTcpClient} returns NULL and records a POSIX error code that can be retrieved with \texttt{Tcl\_GetErrno}. In addition, if interp is non-NULL, an error message is left in the interpreter’s result.

The newly created channel is not registered in the supplied interpreter; to register it, use \texttt{Tcl\_RegisterChannel}. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

\begin{Verbatim}
TCL\_MAKETCPCLIENTCHANNEL
\end{Verbatim}

\texttt{Tcl\_MakeTcpClientChannel} creates a \texttt{Tcl\_Channel} around an existing, platform specific, handle for a client TCP socket.

The newly created channel is not registered in the supplied interpreter; to register it, use \texttt{Tcl\_RegisterChannel}. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of creating the new channel also assigns it as a replacement for the standard channel.

\begin{Verbatim}
TCL\_OPENTCPSERVER
\end{Verbatim}

\texttt{Tcl\_OpenTcpServer} opens a TCP socket on the local host on a specified \textit{port} and uses the Tcl event mechanism to accept requests from clients to connect to it. The \textit{myaddr} argument specifies the network interface. If \textit{myaddr} is NULL the special address INADDR\_ANY should be used to allow connections from any network interface. Each time a client connects to this socket, Tcl creates a channel for the new
connection and invokes proc with information about the channel. Proc must match the following prototype:

```c
typedef void Tcl_TcpAcceptProc(
    ClientData clientData,
    Tcl_Channel channel,
    char *hostName,
    int port);
```

The clientData argument will be the same as the clientData argument to Tcl_OpenTcpServer, channel will be the handle for the new channel, hostName points to a string containing the name of the client host making the connection, and port will contain the client's port number. The new channel is opened for both input and output. If proc raises an error, the connection is closed automatically. Proc has no return value, but if it wishes to reject the connection it can close channel.

Tcl_OpenTcpServer normally returns a pointer to a channel representing the server socket. If an error occurs, Tcl_OpenTcpServer returns NULL and records a POSIX error code that can be retrieved with Tcl_GetErrno. In addition, if the interpreter is non-NULL, an error message is left in the interpreter's result.

The channel returned by Tcl_OpenTcpServer cannot be used for either input or output. It is simply a handle for the socket used to accept connections. The caller can close the channel to shut down the server and disallow further connections from new clients.

TCP server channels operate correctly only in applications that dispatch events through Tcl_DoOneEvent or through Tcl commands such as vwait; otherwise Tcl will never notice that a connection request from a remote client is pending.

The newly created channel is not registered in the supplied interpreter; to register it, use Tcl_RegisterChannel. If one of the standard channels, stdin, stdout or stderr was previously closed, the act of
creating the new channel also assigns it as a replacement for the standard channel.

**PLATFORM ISSUES**

On Unix platforms, the socket handle is a Unix file descriptor as returned by the `socket` system call. On the Windows platform, the socket handle is a `SOCKET` as defined in the WinSock API.

**SEE ALSO**

Tcl.SendMessage, Tcl_RegisterChannel, vwait

**KEYWORDS**

client, server, TCP
NAME
Tcl_CreateEventSource, Tcl_DeleteEventSource, Tcl_SetMaxBlockTime, Tcl_QueueEvent, Tcl_ThreadQueueEvent, Tcl_ThreadAlert, Tcl_GetCurrentThread, Tcl_DeleteEvents, Tcl_InitNotifier, Tcl_FinalizeNotifier, Tcl_WaitForEvent, Tcl_AlertNotifier, Tcl_SetTimer, Tcl_ServiceAll, Tcl_ServiceEvent, Tcl_GetServiceMode, Tcl_SetServiceMode - the event queue and notifier interfaces

SYNOPSIS
#include <tcl.h>
void
Tcl_CreateEventSource(setupProc, checkProc, clientData)
void
Tcl_DeleteEventSource(setupProc, checkProc, clientData)
void
Tcl_SetMaxBlockTime(timePtr)
void
Tcl_QueueEvent(evPtr, position)
void
Tcl_ThreadQueueEvent(threadId, evPtr, position)
void
Tcl_ThreadAlert(threadId)
Tcl_ThreadId
Tcl_GetCurrentThread()
void
Tcl_DeleteEvents(deleteProc, clientData)
ClientData
Tcl_InitNotifier()
void
Tcl_FinalizeNotifier(clientData)
int
Tcl_WaitForEvent(timePtr)
void
Tcl_AlertNotifier(clientData)
void
Tcl_SetTimer(timePtr)
int
Tcl_ServiceAll()
int
Tcl_ServiceEvent(flags)
int
Tcl_GetServiceMode()
int
Tcl_SetServiceMode(mode)
void
Tcl_ServiceModeHook(mode)
void
Tcl_SetNotifier(notifierProcPtr)

ARGUMENTS

INTRODUCTION

NOTIFIER BASICS

CREATING A NEW EVENT SOURCE
  TCL_QUEUE_TAIL
  TCL_QUEUE_HEAD
  TCL_QUEUE_MARK

CREATING A NEW NOTIFIER

REPLACING THE NOTIFIER

EXTERNAL EVENT LOOPS

SEE ALSO

KEYWORDS

NAME

Tcl_CreateEventSource, Tcl_DeleteEventSource,
Tcl_SetMaxBlockTime, Tcl_QueueEvent, Tcl_ThreadQueueEvent,
Tcl_ThreadAlert, Tcl_GetCurrentThread, Tcl_DeleteEvents,
Tcl_InitNotifier, Tcl_FinalizeNotifier, Tcl_WaitForEvent, Tcl_AlertNotifier,
Tcl_SetTimer, Tcl_ServiceAll, Tcl_ServiceEvent, Tcl_GetServiceMode, Tcl_SetServiceMode - the event queue and notifier interfaces

**SYNOPSIS**

```c
#include <tcl.h>
void Tcl_CreateEventSource(setupProc, checkProc, clientData)
void Tcl_DeleteEventSource(setupProc, checkProc, clientData)
void Tcl_SetMaxBlockTime(timePtr)
void Tcl_QueueEvent(evPtr, position)
void Tcl_ThreadQueueEvent(threadId, evPtr, position)
void Tcl_ThreadAlert(threadId)
TclThreadId Tcl_GetCurrentThread()
void Tcl_DeleteEvents(deleteProc, clientData)
ClientData Tcl_InitNotifier()
void Tcl_FinalizeNotifier(clientData)
int Tcl_WaitForEvent(timePtr)
void Tcl_AlertNotifier(clientData)
void Tcl_SetTimer(timePtr)
int Tcl_ServiceAll()
int Tcl_ServiceEvent(flags)
```
ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_EventSetupProc</strong> <em>setupProc</em> (in)</td>
<td>Procedure to invoke to prepare for event wait in <strong>Tcl_DoOneEvent</strong>.</td>
</tr>
<tr>
<td><strong>Tcl_EventCheckProc</strong> <em>checkProc</em> (in)</td>
<td>Procedure for <strong>Tcl_DoOneEvent</strong> to invoke after waiting for events. Checks to see if any events have occurred and, if so, queues them.</td>
</tr>
<tr>
<td><strong>ClientData</strong> <em>clientData</em> (in)</td>
<td>Arbitrary one-word value to pass to <strong>setupProc</strong>, <strong>checkProc</strong>, or <strong>deleteProc</strong>.</td>
</tr>
<tr>
<td><strong>Tcl_Time</strong> <em>timePtr</em> (in)</td>
<td>Indicates the maximum amount of time to wait for an event. This is specified as an interval (how long to wait), not an absolute time (when to wakeup). If the pointer passed to <strong>Tcl_WaitForEvent</strong> is NULL, it means there is no maximum wait time: wait forever if necessary.</td>
</tr>
<tr>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>*<em>Tcl_Event <em>evPtr</em></em> (in)</td>
<td>An event to add to the event queue. The storage for the event must have been allocated by the caller using <code>Tcl_Alloc</code> or <code>ckalloc</code>.</td>
</tr>
<tr>
<td><strong>Tcl_QueuePosition position</strong> (in)</td>
<td>Where to add the new event in the queue: <code>TCL_QUEUE_TAIL</code>, <code>TCL_QUEUE_HEAD</code>, or <code>TCL_QUEUE_MARK</code>.</td>
</tr>
<tr>
<td><strong>Tcl_Threadld threadld</strong> (in)</td>
<td>A unique identifier for a thread.</td>
</tr>
<tr>
<td>*<em>Tcl_EventDeleteProc <em>deleteProc</em></em> (in)</td>
<td>Procedure to invoke for each queued event in <code>Tcl_DeleteEvents</code>.</td>
</tr>
<tr>
<td><strong>int flags</strong> (in)</td>
<td>What types of events to service. These flags are the same as those passed to <code>Tcl_DoOneEvent</code>.</td>
</tr>
<tr>
<td><strong>int mode</strong> (in)</td>
<td>Indicates whether events should be serviced by <code>Tcl_ServiceAll</code>. Must be one of <code>TCL_SERVICE_NONE</code> or <code>TCL_SERVICE_ALL</code>.</td>
</tr>
<tr>
<td><em><em>Tcl_NotifierProcs</em> notifierProcPtr</em>* (in)</td>
<td>Structure of function pointers describing notifier procedures that are to replace the ones installed in the executable. See</td>
</tr>
</tbody>
</table>
INTRODUCTION

The interfaces described here are used to customize the Tcl event loop. The two most common customizations are to add new sources of events and to merge Tcl's event loop with some other event loop, such as one provided by an application in which Tcl is embedded. Each of these tasks is described in a separate section below.

The procedures in this manual entry are the building blocks out of which the Tcl event notifier is constructed. The event notifier is the lowest layer in the Tcl event mechanism. It consists of three things:

[1] Event sources: these represent the ways in which events can be generated. For example, there is a timer event source that implements the `Tcl_CreateTimerHandler` procedure and the `after` command, and there is a file event source that implements the `Tcl_CreateFileHandler` procedure on Unix systems. An event source must work with the notifier to detect events at the right times, record them on the event queue, and eventually notify higher-level software that they have occurred. The procedures `Tcl_CreateEventSource`, `Tcl_DeleteEventSource`, and `Tcl_SetMaxBlockTime`, `Tcl_QueueEvent`, and `Tcl_DeleteEvents` are used primarily by event sources.

[2] The event queue: for non-threaded applications, there is a single queue for the whole application, containing events that have been detected but not yet serviced. Event sources place events onto the queue so that they may be processed in order at appropriate times during the event loop. The event queue guarantees a fair discipline of event handling, so that no event source can starve the others. It also allows events to be saved for servicing at a future time.Threaded applications work in a similar manner, except that there...
Tcl is a separate event queue for each thread containing a Tcl interpreter. **Tcl.QueueEvent** is used (primarily by event sources) to add events to the event queue and **Tcl_DeleteEvents** is used to remove events from the queue without processing them. In a threaded application, **Tcl.QueueEvent** adds an event to the current thread's queue, and **Tcl_ThreadQueueEvent** adds an event to a queue in a specific thread.

The event loop: in order to detect and process events, the application enters a loop that waits for events to occur, places them on the event queue, and then processes them. Most applications will do this by calling the procedure **Tcl.DoOneEvent**, which is described in a separate manual entry.

Most Tcl applications need not worry about any of the internals of the Tcl notifier. However, the notifier now has enough flexibility to be retargeted either for a new platform or to use an external event loop (such as the Motif event loop, when Tcl is embedded in a Motif application). The procedures **Tcl.WaitForEvent** and **Tcl.SetTimer** are normally implemented by Tcl, but may be replaced with new versions to retarget the notifier (the **Tcl.InitNotifier**, **Tcl.AlertNotifier**, **Tcl.FinalizeNotifier**, **Tcl.Sleep**, **Tcl_CreateFileHandler**, and **Tcl_DeleteFileHandler** must also be replaced; see CREATING A NEW NOTIFIER below for details). The procedures **Tcl_ServiceAll**, **Tcl_ServiceEvent**, **Tcl.GetServiceMode**, and **Tcl_SetServiceMode** are provided to help connect Tcl's event loop to an external event loop such as Motif's.

**NOTIFIER BASICS**

The easiest way to understand how the notifier works is to consider what happens when **Tcl.DoOneEvent** is called. **Tcl.DoOneEvent** is passed a **flags** argument that indicates what sort of events it is OK to process and also whether or not to block if no events are ready. **Tcl.DoOneEvent** does the following things:

[1]
Check the event queue to see if it contains any events that can be serviced. If so, service the first possible event, remove it from the queue, and return. It does this by calling `Tcl_ServiceEvent` and passing in the `flags` argument.

[2] Prepare to block for an event. To do this, `Tcl_DoOneEvent` invokes a setup procedure in each event source. The event source will perform event-source specific initialization and possibly call `Tcl_SetMaxBlockTime` to limit how long `Tcl_WaitForEvent` will block if no new events occur.

[3] Call `Tcl_WaitForEvent`. This procedure is implemented differently on different platforms; it waits for an event to occur, based on the information provided by the event sources. It may cause the application to block if `timePtr` specifies an interval other than 0. `Tcl_WaitForEvent` returns when something has happened, such as a file becoming readable or the interval given by `timePtr` expiring. If there are no events for `Tcl_WaitForEvent` to wait for, so that it would block forever, then it returns immediately and `Tcl_DoOneEvent` returns 0.

[4] Call a check procedure in each event source. The check procedure determines whether any events of interest to this source occurred. If so, the events are added to the event queue.

[5] Check the event queue to see if it contains any events that can be serviced. If so, service the first possible event, remove it from the queue, and return.

[6] See if there are idle callbacks pending. If so, invoke all of them and return.

[7]
Either return 0 to indicate that no events were ready, or go back to step [2] if blocking was requested by the caller.

### CREATING A NEW EVENT SOURCE

An event source consists of three procedures invoked by the notifier, plus additional C procedures that are invoked by higher-level code to arrange for event-driven callbacks. The three procedures called by the notifier consist of the setup and check procedures described above, plus an additional procedure that is invoked when an event is removed from the event queue for servicing.

The procedure **Tcl_CreateEventSource** creates a new event source. Its arguments specify the setup procedure and check procedure for the event source. **SetupProc** should match the following prototype:

```c
typedef void Tcl_EventSetupProc(
    ClientData clientData,
    int flags);
```

The **clientData** argument will be the same as the **clientData** argument to **Tcl_CreateEventSource**; it is typically used to point to private information managed by the event source. The **flags** argument will be the same as the **flags** argument passed to **Tcl_DoOneEvent** except that it will never be 0 (**Tcl_DoOneEvent** replaces 0 with **TCL_ALL_EVENTS**). **Flags** indicates what kinds of events should be considered; if the bit corresponding to this event source is not set, the event source should return immediately without doing anything. For example, the file event source checks for the **TCL_FILE_EVENTS** bit.

**SetupProc**'s job is to make sure that the application wakes up when events of the desired type occur. This is typically done in a platform-dependent fashion. For example, under Unix an event source might call **Tcl_CreateFileHandler**; under Windows it might request notification with a Windows event. For timer-driven event sources such as timer events or any polled event, the event source can call
**Tcl_SetMaxBlockTime** to force the application to wake up after a specified time even if no events have occurred. If no event source calls **Tcl_SetMaxBlockTime** then **Tcl_WaitForEvent** will wait as long as necessary for an event to occur; otherwise, it will only wait as long as the shortest interval passed to **Tcl_SetMaxBlockTime** by one of the event sources. If an event source knows that it already has events ready to report, it can request a zero maximum block time. For example, the setup procedure for the X event source looks to see if there are events already queued. If there are, it calls **Tcl_SetMaxBlockTime** with a 0 block time so that **Tcl_WaitForEvent** does not block if there is no new data on the X connection. The **timePtr** argument to **Tcl_WaitForEvent** points to a structure that describes a time interval in seconds and microseconds:

```c
typedef struct Tcl_Time {
    long sec;
    long usec;
} Tcl_Time;
```

The **usec** field should be less than 1000000.

Information provided to **Tcl_SetMaxBlockTime** is only used for the next call to **Tcl_WaitForEvent**; it is discarded after **Tcl_WaitForEvent** returns. The next time an event wait is done each of the event sources' setup procedures will be called again, and they can specify new information for that event wait.

If the application uses an external event loop rather than **Tcl_DoOneEvent**, the event sources may need to call **Tcl_SetMaxBlockTime** at other times. For example, if a new event handler is registered that needs to poll for events, the event source may call **Tcl_SetMaxBlockTime** to set the block time to zero to force the external event loop to call Tcl. In this case, **Tcl_SetMaxBlockTime** invokes **Tcl_SetTimer** with the shortest interval seen since the last call to **Tcl_DoOneEvent** or **Tcl_ServiceAll**.
In addition to the generic procedure `Tcl_SetMaxBlockTime`, other platform-specific procedures may also be available for `setupProc`, if there is additional information needed by `Tcl_WaitForEvent` on that platform. For example, on Unix systems the `Tcl_CreateFileHandler` interface can be used to wait for file events.

The second procedure provided by each event source is its check procedure, indicated by the `checkProc` argument to `Tcl_CreateEventSource`. `CheckProc` must match the following prototype:

```c
typedef void Tcl_EventCheckProc(
    ClientData clientData,
    int flags);
```

The arguments to this procedure are the same as those for `setupProc`. `CheckProc` is invoked by `Tcl_DoOneEvent` after it has waited for events. Presumably at least one event source is now prepared to queue an event. `Tcl_DoOneEvent` calls each of the event sources in turn, so they all have a chance to queue any events that are ready. The check procedure does two things. First, it must see if any events have triggered. Different event sources do this in different ways.

If an event source's check procedure detects an interesting event, it must add the event to Tcl's event queue. To do this, the event source calls `Tcl_QueueEvent`. The `evPtr` argument is a pointer to a dynamically allocated structure containing the event (see below for more information on memory management issues). Each event source can define its own event structure with whatever information is relevant to that event source. However, the first element of the structure must be a structure of type `Tcl_Event`, and the address of this structure is used when communicating between the event source and the rest of the notifier. A `Tcl_Event` has the following definition:

```c
typedef struct {
```
The event source must fill in the `proc` field of the event before calling `Tcl_QueueEvent`. The `nextPtr` is used to link together the events in the queue and should not be modified by the event source.

An event may be added to the queue at any of three positions, depending on the `position` argument to `Tcl_QueueEvent`:

**TCL_QUEUE_TAIL**
Add the event at the back of the queue, so that all other pending events will be serviced first. This is almost always the right place for new events.

**TCL_QUEUE_HEAD**
Add the event at the front of the queue, so that it will be serviced before all other queued events.

**TCL_QUEUE_MARK**
Add the event at the front of the queue, unless there are other events at the front whose position is `TCL_QUEUE_MARK`; if so, add the new event just after all other `TCL_QUEUE_MARK` events. This value of `position` is used to insert an ordered sequence of events at the front of the queue, such as a series of Enter and Leave events synthesized during a grab or ungrab operation in Tk.

When it is time to handle an event from the queue (steps 1 and 4 above) `Tcl_ServiceEvent` will invoke the `proc` specified in the first queued `Tcl_Event` structure. `Proc` must match the following prototype:

```c
typedef int Tcl_EventProc(
    Tcl_Event *evPtr,
    int flags);
```
The first argument to *proc* is a pointer to the event, which will be the same as the first argument to the *Tcl_QueueEvent* call that added the event to the queue. The second argument to *proc* is the *flags* argument for the current call to *Tcl_ServiceEvent*; this is used by the event source to return immediately if its events are not relevant.

It is up to *proc* to handle the event, typically by invoking one or more Tcl commands or C-level callbacks. Once the event source has finished handling the event it returns 1 to indicate that the event can be removed from the queue. If for some reason the event source decides that the event cannot be handled at this time, it may return 0 to indicate that the event should be deferred for processing later; in this case *Tcl_ServiceEvent* will go on to the next event in the queue and attempt to service it. There are several reasons why an event source might defer an event. One possibility is that events of this type are excluded by the *flags* argument. For example, the file event source will always return 0 if the *TCL_FILE_EVENTS* bit is not set in *flags*. Another example of deferring events happens in Tk if *Tk_RestrictEvents* has been invoked to defer certain kinds of window events.

When *proc* returns 1, *Tcl_ServiceEvent* will remove the event from the event queue and free its storage. Note that the storage for an event must be allocated by the event source (using *Tcl_Alloc* or the Tcl macro *ckalloc*) before calling *Tcl_QueueEvent*, but it will be freed by *Tcl_ServiceEvent*, not by the event source.

Threaded applications work in a similar manner, except that there is a separate event queue for each thread containing a Tcl interpreter. Calling *Tcl_QueueEvent* in a multithreaded application adds an event to the current thread's queue. To add an event to another thread's queue, use *Tcl_ThreadQueueEvent*. *Tcl_ThreadQueueEvent* accepts as an argument a *Tcl_ThreadId* argument, which uniquely identifies a thread in a Tcl application. To obtain the *Tcl_ThreadID* for the current thread, use the *Tcl_GetCurrentThread* procedure. (A thread would then need to pass this identifier to other threads for those threads to be able to add events to its queue.) After adding an event to another thread's queue, you then typically need to call *Tcl_ThreadAlert* to “wake up” that thread's notifier to alert it to the new event.
**Tcl_DeleteEvents** can be used to explicitly remove one or more events from the event queue. **Tcl_DeleteEvents** calls *proc* for each event in the queue, deleting those for which the procedure returns 1. Events for which the procedure returns 0 are left in the queue. *Proc* should match the following prototype:

```c
typedef int Tcl_EventDeleteProc(  
    Tcl_Event *evPtr,  
    ClientData clientData);
```

The *clientData* argument will be the same as the *clientData* argument to **Tcl_DeleteEvents**; it is typically used to point to private information managed by the event source. The *evPtr* will point to the next event in the queue.

**Tcl_DeleteEventSource** deletes an event source. The *setupProc*, *checkProc*, and *clientData* arguments must exactly match those provided to the **Tcl_CreateEventSource** for the event source to be deleted. If no such source exists, **Tcl_DeleteEventSource** has no effect.

---

**CREATING A NEW NOTIFIER**

The notifier consists of all the procedures described in this manual entry, plus **Tcl_DoOneEvent** and **Tcl_Sleep**, which are available on all platforms, and **Tcl_CreateFileHandler** and **Tcl_DeleteFileHandler**, which are Unix-specific. Most of these procedures are generic, in that they are the same for all notifiers. However, none of the procedures are notifier-dependent: **Tcl_InitNotifier**, **Tcl_AlertNotifier**, **Tcl_FinalizeNotifier**, **Tcl_SetTimer**, **Tcl_Sleep**, **Tcl_WaitForEvent**, **Tcl_CreateFileHandler**, **Tcl_DeleteFileHandler** and **Tcl_ServiceModeHook**. To support a new platform or to integrate Tcl with an application-specific event loop, you must write new versions of these procedures.
**Tcl_InitNotifier** initializes the notifier state and returns a handle to the notifier state. Tcl calls this procedure when initializing a Tcl interpreter. Similarly, **Tcl_FinalizeNotifier** shuts down the notifier, and is called by **Tcl_Finalize** when shutting down a Tcl interpreter.

**Tcl_WaitForEvent** is the lowest-level procedure in the notifier; it is responsible for waiting for an “interesting” event to occur or for a given time to elapse. Before **Tcl_WaitForEvent** is invoked, each of the event sources' setup procedure will have been invoked. The *timePtr* argument to **Tcl_WaitForEvent** gives the maximum time to block for an event, based on calls to **Tcl_SetMaxBlockTime** made by setup procedures and on other information (such as the **TCL_DONT_WAIT** bit in *flags*).

Ideally, **Tcl_WaitForEvent** should only wait for an event to occur; it should not actually process the event in any way. Later on, the event sources will process the raw events and create Tcl_Events on the event queue in their *checkProc* procedures. However, on some platforms (such as Windows) this is not possible; events may be processed in **Tcl_WaitForEvent**, including queuing Tcl_Events and more (for example, callbacks for native widgets may be invoked). The return value from **Tcl_WaitForEvent** must be either 0, 1, or -1. On platforms such as Windows where events get processed in **Tcl_WaitForEvent**, a return value of 1 means that there may be more events still pending that have not been processed. This is a sign to the caller that it must call **Tcl_WaitForEvent** again if it wants all pending events to be processed. A 0 return value means that calling **Tcl_WaitForEvent** again will not have any effect: either this is a platform where **Tcl_WaitForEvent** only waits without doing any event processing, or **Tcl_WaitForEvent** knows for sure that there are no additional events to process (e.g. it returned because the time elapsed). Finally, a return value of -1 means that the event loop is no longer operational and the application should probably unwind and terminate. Under Windows this happens when a WM_QUIT message is received; under Unix it happens when **Tcl_WaitForEvent** would have waited forever because there were no active event sources and the timeout was infinite.

**Tcl_AlertNotifier** is used in multithreaded applications to allow any thread to “wake up” the notifier to alert it to new events on its queue.
Tcl_AlertNotifier requires as an argument the notifier handle returned by Tcl_InitNotifier.

If the notifier will be used with an external event loop, then it must also support the Tcl_SetTimer interface. Tcl_SetTimer is invoked by Tcl_SetMaxBlockTime whenever the maximum blocking time has been reduced. Tcl_SetTimer should arrange for the external event loop to invoke Tcl_ServiceAll after the specified interval even if no events have occurred. This interface is needed because Tcl_WaitForEvent is not invoked when there is an external event loop. If the notifier will only be used from Tcl_DoOneEvent, then Tcl_SetTimer need not do anything.

Tcl_ServiceModeHook is called by the platform-independent portion of the notifier when client code makes a call to Tcl_SetServiceMode. This hook is provided to support operating systems that require special event handling when the application is in a modal loop (the Windows notifier, for instance, uses this hook to create a communication window).

On Unix systems, the file event source also needs support from the notifier. The file event source consists of the Tcl_CreateFileHandler and Tcl_DeleteFileHandler procedures, which are described in the Tcl_CreateFileHandler manual page.

The Tcl_Sleep and Tcl_DoOneEvent interfaces are described in their respective manual pages.

The easiest way to create a new notifier is to look at the code for an existing notifier, such as the files unix/tclUnixNotfy.c or win/tclWinNotify.c in the Tcl source distribution.

REPLACING THE NOTIFIER

A notifier that has been written according to the conventions above can also be installed in a running process in place of the standard notifier. This mechanism is used so that a single executable can be used (with the standard notifier) as a stand-alone program and reused (with a replacement notifier in a loadable extension) as an extension to another
program, such as a Web browser plugin.

To do this, the extension makes a call to `Tcl_SetNotifier` passing a pointer to a `Tcl_NotifierProcs` data structure. The structure has the following layout:

```c
typedef struct Tcl_NotifierProcs {
    Tcl_SetTimerProc *setTimerProc;
    Tcl_WaitForEventProc *waitForEventProc;
    Tcl_CreateFileHandlerProc *createFileHandlerProc;
    Tcl_DeleteFileHandlerProc *deleteFileHandlerProc;
    Tcl_InitNotifierProc *initNotifierProc;
    Tcl_FinalizeNotifierProc *finalizeNotifierProc;
    Tcl_AlertNotifierProc *alertNotifierProc;
    Tcl_ServiceModeHookProc *serviceModeHookProc;
} Tcl_NotifierProcs;
```

Following the call to `Tcl_SetNotifier`, the pointers given in the `Tcl_NotifierProcs` structure replace whatever notifier had been installed in the process.

It is extraordinarily unwise to replace a running notifier. Normally, `Tcl_SetNotifier` should be called at process initialization time before the first call to `Tcl_InitNotifier`.

### EXTERNAL EVENT LOOPS

The notifier interfaces are designed so that Tcl can be embedded into applications that have their own private event loops. In this case, the application does not call `Tcl_DoOneEvent` except in the case of recursive event loops such as calls to the Tcl commands `update` or `vwait`. Most of the time is spent in the external event loop of the application. In this case the notifier must arrange for the external event loop to call back into Tcl when something happens on the various Tcl event sources. These callbacks should arrange for appropriate Tcl events to be placed on the Tcl event queue.
Because the external event loop is not calling **Tcl_DoOneEvent** on a regular basis, it is up to the notifier to arrange for **Tcl_ServiceEvent** to be called whenever events are pending on the Tcl event queue. The easiest way to do this is to invoke **Tcl_ServiceAll** at the end of each callback from the external event loop. This will ensure that all of the event sources are polled, any queued events are serviced, and any pending idle handlers are processed before returning control to the application. In addition, event sources that need to poll for events can call **Tcl_SetMaxBlockTime** to force the external event loop to call Tcl even if no events are available on the system event queue.

As a side effect of processing events detected in the main external event loop, Tcl may invoke **Tcl_DoOneEvent** to start a recursive event loop in commands like **vwait**. **Tcl_DoOneEvent** will invoke the external event loop, which will result in callbacks as described in the preceding paragraph, which will result in calls to **Tcl_ServiceAll**. However, in these cases it is undesirable to service events in **Tcl_ServiceAll**. Servicing events there is unnecessary because control will immediately return to the external event loop and hence to **Tcl_DoOneEvent**, which can service the events itself. Furthermore, **Tcl_DoOneEvent** is supposed to service only a single event, whereas **Tcl_ServiceAll** normally services all pending events. To handle this situation, **Tcl_DoOneEvent** sets a flag for **Tcl_ServiceAll** that causes it to return without servicing any events. This flag is called the service mode; **Tcl_DoOneEvent** restores it to its previous value before it returns.

In some cases, however, it may be necessary for **Tcl_ServiceAll** to service events even when it has been invoked from **Tcl_DoOneEvent**. This happens when there is yet another recursive event loop invoked via an event handler called by **Tcl_DoOneEvent** (such as one that is part of a native widget). In this case, **Tcl_DoOneEvent** may not have a chance to service events so **Tcl_ServiceAll** must service them all. Any recursive event loop that calls an external event loop rather than **Tcl_DoOneEvent** must reset the service mode so that all events get processed in **Tcl_ServiceAll**. This is done by invoking the **Tcl_SetServiceMode** procedure. If **Tcl_SetServiceMode** is passed **TCL_SERVICE_NONE**, then calls to **Tcl_ServiceAll** will return
immediately without processing any events. If `Tcl_SetServiceMode` is passed `TCL_SERVICE_ALL`, then calls to `Tcl_ServiceAll` will behave normally. `Tcl_SetServiceMode` returns the previous value of the service mode, which should be restored when the recursive loop exits. `Tcl_GetServiceMode` returns the current value of the service mode.

**SEE ALSO**

`Tcl_CreateFileHandler`, `Tcl_DeleteFileHandler`, `Tcl_Sleep`, `Tcl_DoOneEvent`, `Thread`

**KEYWORDS**

`event`, `notifier`, `event queue`, `event sources`, `file events`, `timer`, `idle`, `service mode`, `threads`
NAME
Tcl_Panic, Tcl_PanicVA, Tcl_SetPanicProc - report fatal error and abort

SYNOPSIS
#include <tcl.h>
void
Tcl_Panic(format, arg, arg, ...)
void
Tcl_PanicVA(format, argList)
void
Tcl_SetPanicProc(panicProc)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS
DESCRIPTION

When the Tcl library detects that its internal data structures are in an inconsistent state, or that its C procedures have been called in a manner inconsistent with their documentation, it calls Tcl_Panic to display a message describing the error and abort the process. The format argument is a format string describing how to format the remaining arguments arg into an error message, according to the same formatting rules used by the printf family of functions. The same formatting rules are also used by the built-in Tcl command format.

In a freshly loaded Tcl library, Tcl_Panic prints the formatted error message to the standard error file of the process, and then calls abort to terminate the process. Tcl_Panic does not return.

Tcl_SetPanicProc may be used to modify the behavior of Tcl_Panic. The panicProc argument should match the type Tcl_PanicProc:

```c
typedef void Tcl_PanicProc(
    const char *format,  // A printf-style format string.
    arg, arg,...);      // Arguments matching the format string.
```
After `Tcl_SetPanicProc` returns, any future calls to `Tcl_Panic` will call `panicProc`, passing along the `format` and `arg` arguments. To maintain consistency with the callers of `Tcl_Panic`, `panicProc` must not return; it must call `abort`. `panicProc` should avoid making calls into the Tcl library, or into other libraries that may call the Tcl library, since the original call to `Tcl_Panic` indicates the Tcl library is not in a state of reliable operation.

The typical use of `Tcl_SetPanicProc` arranges for the error message to be displayed or reported in a manner more suitable for the application or the platform. As an example, the Windows implementation of `wish` calls `Tcl_SetPanicProc` to force all panic messages to be displayed in a system dialog box, rather than to be printed to the standard error file (usually not visible under Windows).

Although the primary callers of `Tcl_Panic` are the procedures of the Tcl library, `Tcl_Panic` is a public function and may be called by any extension or application that wishes to abort the process and have a panic message displayed the same way that panic messages from Tcl will be displayed.

`Tcl_PanicVA` is the same as `Tcl_Panic` except that instead of taking a variable number of arguments it takes an argument list.

SEE ALSO

`abort`, `printf`, `exec`, `format`

KEYWORDS

`abort`, `fatal`, `error`

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

Tcl_AllowExceptions - allow all exceptions in next script evaluation

SYNOPSIS

#include <tcl.h>
Tcl_AllowExceptions(interp)

ARGUMENTS

Tcl_interp *interp (in)  Interpreter in which script will be evaluated.

DESCRIPTION

If a script is evaluated at top-level (i.e. no other scripts are pending evaluation when the script is invoked), and if the script terminates with a completion code other than TCL_OK, TCL_ERROR or TCL_RETURN, then Tcl normally converts this into a TCL_ERROR return with an appropriate message. The particular script evaluation procedures of Tcl that act in the manner are Tcl_EvalObjEx, Tcl_EvalObjv, Tcl_Eval, Tcl_EvalEx, Tcl_GlobalEval, Tcl_GlobalEvalObj, Tcl_VarEval and Tcl_VarEvalVA.

However, if Tcl_AllowExceptions is invoked immediately before calling one of those a procedures, then arbitrary completion codes are permitted from the script, and they are returned without modification. This is useful in cases where the caller can deal with exceptions such as TCL_BREAK or TCL_CONTINUE in a meaningful way.
<table>
<thead>
<tr>
<th>KEYWORDS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>continue, break, exception, interpreter</td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces,
Tcl_ParseQuotedString, Tcl_ParseVarName, Tcl_ParseVar,
Tcl_FreeParse, Tcl_EvalTokens, Tcl_EvalTokensStandard -
parse Tcl scripts and expressions

SYNOPSIS
#include <tcl.h>
int
Tcl_ParseCommand(interp, start, numBytes, nested,
parsePtr)
int
Tcl_ParseExpr(interp, start, numBytes, parsePtr)
int
Tcl_ParseBraces(interp, start, numBytes, parsePtr, append,
termPtr)
int
Tcl_ParseQuotedString(interp, start, numBytes, parsePtr,
append, termPtr)
int
Tcl_ParseVarName(interp, start, numBytes, parsePtr, append)
const char *
Tcl_ParseVar(interp, start, termPtr)
Tcl_FreeParse(usedParsePtr)
Tcl_Obj *
Tcl_EvalTokens(interp, tokenPtr, numTokens)
int
Tcl_EvalTokensStandard(interp, tokenPtr, numTokens)

ARGUMENTS
DESCRIPTION
TCL_PARSE STRUCTURE
TCL_TOKEN_WORD
Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces,
Tcl_ParseQuotedString, Tcl_ParseVarName, Tcl_ParseVar,
Tcl_FreeParse, Tcl_EvalTokens, Tcl_EvalTokensStandard - parse Tcl
scripts and expressions

SYNOPSIS

#include <tcl.h>
int
Tcl_ParseCommand(interp, start, numBytes, nested, parsePtr)
int
Tcl_ParseExpr(interp, start, numBytes, parsePtr)
int
Tcl_ParseBraces(interp, start, numBytes, parsePtr, append, termPtr)
int
Tcl_ParseQuotedString(interp, start, numBytes, parsePtr, append, termPtr)
int
Tcl_ParseVarName(interp, start, numBytes, parsePtr, append)
const char *
Tcl_ParseVar(interp, start, termPtr)
Tcl_FreeParse(usedParsePtr)
Tcl_Obj *
Tcl_EvalTokens(interp, tokenPtr, numTokens)
int
Tcl_EvalTokensStandard(interp, tokenPtr, numTokens)
### ARGUMENTS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp</em> (out)</td>
<td>For procedures other than <strong>Tcl_FreeParse</strong>, <strong>Tcl_EvalTokens</strong> and <strong>Tcl_EvalTokensStandard</strong>, used only for error reporting; if NULL, then no error messages are left after errors. For <strong>Tcl_EvalTokens</strong> and <strong>Tcl_EvalTokensStandard</strong>, determines the context for evaluating the script and also is used for error reporting; must not be NULL.</td>
</tr>
<tr>
<td>const char <em>start</em> (in)</td>
<td>Pointer to first character in string to parse.</td>
</tr>
<tr>
<td>int <strong>numBytes</strong> (in)</td>
<td>Number of bytes in string to parse, not including any terminating null character. If less than 0 then the script consists of all characters following <em>start</em> up to the first null character.</td>
</tr>
<tr>
<td>int <strong>nested</strong> (in)</td>
<td>Non-zero means that the script is part of a command substitution so an unquoted close bracket should be treated as a command terminator. If</td>
</tr>
</tbody>
</table>
zero, close brackets have no special meaning.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <strong>append</strong></td>
<td>(in) Non-zero means that *parsePtr already contains valid tokens; the new tokens should be appended to those already present. Zero means that *parsePtr is uninitialized; any information in it is ignored. This argument is normally 0.</td>
</tr>
<tr>
<td>Tcl_Parse <strong>parsePtr</strong> (out)</td>
<td>Points to structure to fill in with information about the parsed command, expression, variable name, etc. Any previous information in this structure is ignored, unless **append is non-zero in a call to Tcl_ParseBraces, Tcl_ParseQuotedString, or Tcl_ParseVarName.</td>
</tr>
<tr>
<td>const char <strong>termPtr</strong> (out)</td>
<td>If not NULL, points to a location where Tcl_ParseBraces, Tcl_ParseQuotedString, and Tcl_ParseVar will store a pointer to the character just after the terminating character (the close-brace, the last character of the variable name, or the close-quote</td>
</tr>
</tbody>
</table>
(respectively)) if the parse was successful.

Tcl_Parse *usedParsePtr (in)  Points to structure that was filled in by a previous call to
Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseVarName, etc.

### DESCRIPTION

These procedures parse Tcl commands or portions of Tcl commands such as expressions or references to variables. Each procedure takes a pointer to a script (or portion thereof) and fills in the structure pointed to by parsePtr with a collection of tokens describing the information that was parsed. The procedures normally return TCL_OK. However, if an error occurs then they return TCL_ERROR, leave an error message in interp's result (if interp is not NULL), and leave nothing in parsePtr.

**Tcl_ParseCommand** is a procedure that parses Tcl scripts. Given a pointer to a script, it parses the first command from the script. If the command was parsed successfully, **Tcl_ParseCommand** returns TCL_OK and fills in the structure pointed to by parsePtr with information about the structure of the command (see below for details). If an error occurred in parsing the command then TCL_ERROR is returned, an error message is left in interp's result, and no information is left at *parsePtr.

**Tcl_ParseExpr** parses Tcl expressions. Given a pointer to a script containing an expression, **Tcl_ParseExpr** parses the expression. If the expression was parsed successfully, **Tcl_ParseExpr** returns TCL_OK and fills in the structure pointed to by parsePtr with information about the structure of the expression (see below for details). If an error occurred in parsing the command then TCL_ERROR is returned, an error message is left in interp's result, and no information is left at *parsePtr.
**Tcl_ParseBraces** parses a string or command argument enclosed in braces such as `{hello}` or `{string \t with \t tabs}` from the beginning of its argument `start`. The first character of `start` must be `{`. If the braced string was parsed successfully, **Tcl_ParseBraces** returns **TCL_OK**, fills in the structure pointed to by `parsePtr` with information about the structure of the string (see below for details), and stores a pointer to the character just after the terminating `}` in the location given by `*termPtr`. If an error occurs while parsing the string then **TCL_ERROR** is returned, an error message is left in `interp`'s result, and no information is left at `*parsePtr` or `*termPtr`.

**Tcl_ParseQuotedString** parses a double-quoted string such as "sum is [expr {$a+$b}]" from the beginning of the argument `start`. The first character of `start` must be ". If the double-quoted string was parsed successfully, **Tcl_ParseQuotedString** returns **TCL_OK**, fills in the structure pointed to by `parsePtr` with information about the structure of the string (see below for details), and stores a pointer to the character just after the terminating " in the location given by `*termPtr`. If an error occurs while parsing the string then **TCL_ERROR** is returned, an error message is left in `interp`'s result, and no information is left at `*parsePtr` or `*termPtr`.

**Tcl_ParseVarName** parses a Tcl variable reference such as `$abc` or `$x([expr {$index + 1}])` from the beginning of its `start` argument. The first character of `start` must be $`. If a variable name was parsed successfully, **Tcl_ParseVarName** returns **TCL_OK** and fills in the structure pointed to by `parsePtr` with information about the structure of the variable name (see below for details). If an error occurs while parsing the command then **TCL_ERROR** is returned, an error message is left in `interp`'s result (if `interp` is not NULL), and no information is left at `*parsePtr`.

**Tcl_ParseVar** parse a Tcl variable reference such as `$abc` or `$x([expr {$index + 1}])` from the beginning of its `start` argument. The first character of `start` must be $`. If the variable name is parsed successfully, **Tcl_ParseVar** returns a pointer to the string value of the variable. If an error occurs while parsing, then NULL is returned and an error message is left in `interp`'s result.
The information left at *parsePtr by Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces, Tcl_ParseQuotedString, and Tcl_ParseVarName may include dynamically allocated memory. If these five parsing procedures return TCL_OK then the caller must invoke Tcl_FreeParse to release the storage at *parsePtr. These procedures ignore any existing information in *parsePtr (unless append is non-zero), so if repeated calls are being made to any of them then Tcl_FreeParse must be invoked once after each call.

**Tcl_EvalTokensStandard** evaluates a sequence of parse tokens from a Tcl_Parse structure. The tokens typically consist of all the tokens in a word or all the tokens that make up the index for a reference to an array variable. **Tcl_EvalTokensStandard** performs the substitutions requested by the tokens and concatenates the resulting values. The return value from **Tcl_EvalTokensStandard** is a Tcl completion code with one of the values TCL_OK, TCL_ERROR, TCL_RETURN, TCL_BREAK, or TCL_CONTINUE, or possibly some other integer value originating in an extension. In addition, a result value or error message is left in interp's result; it can be retrieved using Tcl_GetObjResult.

**Tcl_EvalTokens** differs from **Tcl_EvalTokensStandard** only in the return convention used: it returns the result in a new Tcl_Obj. The reference count of the object returned as result has been incremented, so the caller must invoke **Tcl_DecrRefCount** when it is finished with the object. If an error or other exception occurs while evaluating the tokens (such as a reference to a non-existent variable) then the return value is NULL and an error message is left in interp's result. The use of **Tcl_EvalTokens** is deprecated.

**TCL_PARSE STRUCTURE**

Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces, Tcl_ParseQuotedString, and Tcl_ParseVarName return parse information in two data structures, Tcl_Parse and Tcl_Token:
typedef struct Tcl_Parse {
    const char *commentStart;
    int commentSize;
    const char *commandStart;
    int commandSize;
    int numWords;
    Tcl_Token *tokenPtr;
    int numTokens;
    ...
} Tcl_Parse;

typedef struct Tcl_Token {
    int type;
    const char *start;
    int size;
    int numComponents;
} Tcl_Token;

The first five fields of a Tcl_Parse structure are filled in only by
**Tcl_ParseCommand**. These fields are not used by the other parsing procedures.

**Tcl_ParseCommand** fills in a Tcl_Parse structure with information that describes one Tcl command and any comments that precede the command. If there are comments, the *commentStart* field points to the # character that begins the first comment and *commentSize* indicates the number of bytes in all of the comments preceding the command, including the newline character that terminates the last comment. If the command is not preceded by any comments, *commentSize* is 0.

**Tcl_ParseCommand** also sets the *commandStart* field to point to the first character of the first word in the command (skipping any comments and leading space) and *commandSize* gives the total number of bytes in the command, including the character pointed to by *commandStart* up to and including the newline, close bracket, or semicolon character that terminates the command. The *numWords* field gives the total number of words in the command.
All parsing procedures set the remaining fields, tokenPtr and numTokens. The tokenPtr field points to the first in an array of Tcl_Token structures that describe the components of the entity being parsed. The numTokens field gives the total number of tokens present in the array. Each token contains four fields. The type field selects one of several token types that are described below. The start field points to the first character in the token and the size field gives the total number of characters in the token. Some token types, such as TCL_TOKEN_WORD and TCL_TOKEN_VARIABLE, consist of several component tokens, which immediately follow the parent token; the numComponents field describes how many of these there are. The type field has one of the following values:

**TCL_TOKEN_WORD**
This token ordinarily describes one word of a command but it may also describe a quoted or braced string in an expression. The token describes a component of the script that is the result of concatenating together a sequence of subcomponents, each described by a separate subtoken. The token starts with the first non-blank character of the component (which may be a double-quote or open brace) and includes all characters in the component up to but not including the space, semicolon, close bracket, close quote, or close brace that terminates the component. The numComponents field counts the total number of sub-tokens that make up the word, including sub-tokens of **TCL_TOKEN_VARIABLE** and **TCL_TOKEN_BS** tokens.

**TCL_TOKEN_SIMPLE_WORD**
This token has the same meaning as **TCL_TOKEN_WORD**, except that the word is guaranteed to consist of a single **TCL_TOKEN_TEXT** sub-token. The numComponents field is always 1.

**TCL_TOKEN_EXPAND_WORD**
This token has the same meaning as **TCL_TOKEN_WORD**, except that the command parser notes this word began with the expansion prefix \{'\}, indicating that after substitution, the list value of this word should be expanded to form multiple arguments in command
evaluation. This token type can only be created by
Tcl_ParseCommand.

**TCL_TOKEN_TEXT**
The token describes a range of literal text that is part of a word.
The \textit{numComponents} field is always 0.

**TCL_TOKEN_BS**
The token describes a backslash sequence such as \texttt{\textbackslash n} or \texttt{\textbackslash 0xa3}.
The \textit{numComponents} field is always 0.

**TCL_TOKEN_COMMAND**
The token describes a command whose result must be substituted into the word. The token includes the square brackets that surround the command. The \textit{numComponents} field is always 0 (the nested command is not parsed; call \texttt{Tcl_ParseCommand} recursively if you want to see its tokens).

**TCL_TOKEN_VARIABLE**
The token describes a variable substitution, including the $, variable name, and array index (if there is one) up through the close parenthesis that terminates the index. This token is followed by one or more additional tokens that describe the variable name and array index. If \textit{numComponents} is 1 then the variable is a scalar and the next token is a \texttt{TCL_TOKEN_TEXT} token that gives the variable name. If \textit{numComponents} is greater than 1 then the variable is an array: the first sub-token is a \texttt{TCL_TOKEN_TEXT} token giving the array name and the remaining sub-tokens are \texttt{TCL_TOKEN_TEXT}, \texttt{TCL_TOKEN_BS}, \texttt{TCL_TOKEN_COMMAND}, and \texttt{TCL_TOKEN_VARIABLE} tokens that must be concatenated to produce the array index. The \textit{numComponents} field includes nested sub-tokens that are part of \texttt{TCL_TOKEN_VARIABLE} tokens in the array index.

**TCL_TOKEN_SUB_EXPR**
The token describes one subexpression of an expression (or an entire expression). A subexpression may consist of a value such as an integer literal, variable substitution, or parenthesized
subexpression; it may also consist of an operator and its operands. The token starts with the first non-blank character of the subexpression up to but not including the space, brace, close-paren, or bracket that terminates the subexpression. This token is followed by one or more additional tokens that describe the subexpression. If the first sub-token after the TCL_TOKEN_SUB_EXPR token is a TCL_TOKEN_OPERATOR token, the subexpression consists of an operator and its token operands. If the operator has no operands, the subexpression consists of just the TCL_TOKEN_OPERATOR token. Each operand is described by a TCL_TOKEN_SUB_EXPR token. Otherwise, the subexpression is a value described by one of the token types TCL_TOKEN_WORD, TCL_TOKEN_TEXT, TCL_TOKEN_BS, TCL_TOKEN_COMMAND, TCL_TOKEN_VARIABLE, and TCL_TOKEN_SUB_EXPR. The numComponents field counts the total number of sub-tokens that make up the subexpression; this includes the sub-tokens for any nested TCL_TOKEN_SUB_EXPR tokens.

TCL_TOKEN_OPERATOR
The token describes one operator of an expression such as & & or hypot. A TCL_TOKEN_OPERATOR token is always preceded by a TCL_TOKEN_SUB_EXPR token that describes the operator and its operands; the TCL_TOKEN_SUB_EXPR token's numComponents field can be used to determine the number of operands. A binary operator such as * is followed by two TCL_TOKEN_SUB_EXPR tokens that describe its operands. A unary operator like - is followed by a single TCL_TOKEN_SUB_EXPR token for its operand. If the operator is a math function such as log10, the TCL_TOKEN_OPERATOR token will give its name and the following TCL_TOKEN_SUB_EXPR tokens will describe its operands; if there are no operands (as with rand), no TCL_TOKEN_SUB_EXPR tokens follow. There is one trinary operator, ?, that appears in if-then-else subexpressions such as x?y:z; in this case, the ? TCL_TOKEN_OPERATOR token is followed by three TCL_TOKEN_SUB_EXPR tokens for the operands x, y, and z. The numComponents field for a
TCL_TOKEN_OPERATOR token is always 0.

After Tcl_ParseCommand returns, the first token pointed to by the tokenPtr field of the Tcl_Parse structure always has type TCL_TOKEN_WORD or TCL_TOKEN_SIMPLE_WORD or TCL_TOKEN_EXPAND_WORD. It is followed by the sub-tokens that must be concatenated to produce the value of that word. The next token is the TCL_TOKEN_WORD or TCL_TOKEN_SIMPLE_WORD of TCL_TOKEN_EXPAND_WORD token for the second word, followed by sub-tokens for that word, and so on until all numWords have been accounted for.

After Tcl_ParseExpr returns, the first token pointed to by the tokenPtr field of the Tcl_Parse structure always has type TCL_TOKEN_SUB_EXPR. It is followed by the sub-tokens that must be evaluated to produce the value of the expression. Only the token information in the Tcl_Parse structure is modified: the commentStart, commentSize, commandStart, and commandSize fields are not modified by Tcl_ParseExpr.

After Tcl_ParseBraces returns, the array of tokens pointed to by the tokenPtr field of the Tcl_Parse structure will contain a single TCL_TOKEN_TEXT token if the braced string does not contain any backslash-newlines. If the string does contain backslash-newlines, the array of tokens will contain one or more TCL_TOKEN_TEXT or TCL_TOKEN_BS sub-tokens that must be concatenated to produce the value of the string. If the braced string was just {} (that is, the string was empty), the single TCL_TOKEN_TEXT token will have a size field containing zero; this ensures that at least one token appears to describe the braced string. Only the token information in the Tcl_Parse structure is modified: the commentStart, commentSize, commandStart, and commandSize fields are not modified by Tcl_ParseBraces.

After Tcl_ParseQuotedString returns, the array of tokens pointed to by the tokenPtr field of the Tcl_Parse structure depends on the contents of the quoted string. It will consist of one or more TCL_TOKEN_TEXT, TCL_TOKEN_BS, TCL_TOKEN_COMMAND, and TCL_TOKEN_VARIABLE sub-tokens. The array always contains at
least one token; for example, if the argument start is empty, the array returned consists of a single TCL_TOKEN_TEXT token with a zero size field. Only the token information in the Tcl_Parse structure is modified: the commentStart, commentSize, commandStart, and commandSize fields are not modified.

After Tcl_ParseVarName returns, the first token pointed to by the tokenPtr field of the Tcl_Parse structure always has type TCL_TOKEN_VARIABLE. It is followed by the sub-tokens that make up the variable name as described above. The total length of the variable name is contained in the size field of the first token. As in Tcl_ParseExpr, only the token information in the Tcl_Parse structure is modified by Tcl_ParseVarName: the commentStart, commentSize, commandStart, and commandSize fields are not modified.

All of the character pointers in the Tcl_Parse and Tcl_Token structures refer to characters in the start argument passed to Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces, Tcl_ParseQuotedString, and Tcl_ParseVarName.

There are additional fields in the Tcl_Parse structure after the numTokens field, but these are for the private use of Tcl_ParseCommand, Tcl_ParseExpr, Tcl_ParseBraces, Tcl_ParseQuotedString, and Tcl_ParseVarName; they should not be referenced by code outside of these procedures.

KEYWORDS

backslash substitution, braces, command, expression, parse, token, variable substitution

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997 Sun Microsystems, Inc.
### NAME

Tcl_RegisterObjType, Tcl_GetObjType, Tcl_AppendAllObjTypes, Tcl_ConvertToType - manipulate Tcl object types

### SYNOPSIS

```c
#include <tcl.h>

Tcl_RegisterObjType(typePtr)
Tcl_ObjType *

Tcl_GetObjType(typeName)
int

Tcl_AppendAllObjTypes(interp, objPtr)
int

Tcl_ConvertToType(interp, objPtr, typePtr)
```

### ARGUMENTS

**DESCRIPTION**

### THE TCL_OBJTYPE STRUCTURE

### THE NAME FIELD

### THE SETFROMANYPROC FIELD

### THE UPDATESTRINGPROC FIELD

### THE DUPINTREPPROC FIELD

### THE FREEINTREPPROC FIELD

### SEE ALSO

### KEYWORDS
**Tcl_RegisterObjType**(*typePtr*)

**Tcl_ObjType** *

**Tcl_GetObjType**(*typeName*)

int

**Tcl_AppendAllObjTypes**(*interp*, *objPtr*)

int

**Tcl_ConvertToType**(*interp*, *objPtr*, *typePtr*)

---

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_ObjType *typePtr (in)</td>
<td>Points to the structure containing information about the Tcl object type. This storage must live forever, typically by being statically allocated.</td>
</tr>
<tr>
<td>const char *typeName (in)</td>
<td>The name of a Tcl object type that <strong>Tcl_GetObjType</strong> should look up.</td>
</tr>
<tr>
<td>Tcl_Interp *interp (in)</td>
<td>Interpreter to use for error reporting.</td>
</tr>
<tr>
<td>Tcl_Obj *objPtr (in)</td>
<td>For <strong>Tcl_AppendAllObjTypes</strong>, this points to the object onto which it appends the name of each object type as a list element. For <strong>Tcl_ConvertToType</strong>, this points to an object that must have been the result of a previous call to <strong>Tcl_NewObj</strong>.</td>
</tr>
</tbody>
</table>
The procedures in this man page manage Tcl object types. They are used to register new object types, look up types, and force conversions from one type to another.

**Tcl_RegisterObjType** registers a new Tcl object type in the table of all object types that **Tcl_GetObjType** can look up by name. There are other object types supported by Tcl as well, which Tcl chooses not to register. Extensions can likewise choose to register the object types they create or not. The argument `typePtr` points to a `Tcl_ObjType` structure that describes the new type by giving its name and by supplying pointers to four procedures that implement the type. If the type table already contains a type with the same name as in `typePtr`, it is replaced with the new type. The `Tcl_ObjType` structure is described in the section **THE TCL_OBJTYPE STRUCTURE** below.

**Tcl_GetObjType** returns a pointer to the registered `Tcl_ObjType` with name `typeName`. It returns NULL if no type with that name is registered.

**Tcl_AppendAllObjTypes** appends the name of each registered object type as a list element onto the Tcl object referenced by `objPtr`. The return value is **TCL_OK** unless there was an error converting `objPtr` to a list object; in that case **TCL_ERROR** is returned.

**Tcl_ConvertToType** converts an object from one type to another if possible. It creates a new internal representation for `objPtr` appropriate for the target type `typePtr` and sets its `typePtr` member as determined by calling the `typePtr->setFromAnyProc` routine. Any internal representation for `objPtr`'s old type is freed. If an error occurs during conversion, it returns **TCL_ERROR** and leaves an error message in the result object for `interp` unless `interp` is NULL. Otherwise, it returns **TCL_OK**. Passing a NULL `interp` allows this procedure to be used as a test whether the conversion can be done (and in fact was done).

In many cases, the `typePtr->setFromAnyProc` routine will set `objPtr->typePtr` to the argument value `typePtr`, but that is no longer guaranteed. The `setFromAnyProc` is free to set the internal
representation for objPtr to make use of another related Tcl_ObjType, if it sees fit.

**THE TCL_OBJTYPE STRUCTURE**

Extension writers can define new object types by defining four procedures and initializing a Tcl_ObjType structure to describe the type. Extension writers may also pass a pointer to their Tcl_ObjType structure to `Tcl_RegisterObjType` if they wish to permit other extensions to look up their Tcl_ObjType by name with the `Tcl_GetObjType` routine. The `Tcl_ObjType` structure is defined as follows:

```c
typedef struct Tcl_ObjType {
    char *name;
    Tcl_FreeInternalRepProc *freeIntRepProc;
    Tcl_DupInternalRepProc *dupIntRepProc;
    Tcl_UpdateStringProc *updateStringProc;
    Tcl_SetFromAnyProc *setFromAnyProc;
} Tcl_ObjType;
```

**THE NAME FIELD**

The `name` member describes the name of the type, e.g. `int`. When a type is registered, this is the name used by callers of `Tcl_GetObjType` to lookup the type. For unregistered types, the `name` field is primarily of value for debugging. The remaining four members are pointers to procedures called by the generic Tcl object code:

**THE SETFROMANYPROC FIELD**

The `setFromAnyProc` member contains the address of a function called to create a valid internal representation from an object's string representation.

```c
typedef int (Tcl_SetFromAnyProc) (Tcl_Interp *interp, ...
```
If an internal representation cannot be created from the string, it returns **TCL_ERROR** and puts a message describing the error in the result object for *interp* unless *interp* is NULL. If *setFromAnyProc* is successful, it stores the new internal representation, sets *objPtr*'s *typePtr* member to point to the **Tcl_ObjType** struct corresponding to the new internal representation, and returns **TCL_OK**. Before setting the new internal representation, the *setFromAnyProc* must free any internal representation of *objPtr*'s old type; it does this by calling the old type's *freeIntRepProc* if it is not NULL.

As an example, the *setFromAnyProc* for the built-in Tcl list type gets an up-to-date string representation for *objPtr* by calling **Tcl_GetStringFromObj**. It parses the string to verify it is in a valid list format and to obtain each element value in the list, and, if this succeeds, stores the list elements in *objPtr*'s internal representation and sets *objPtr*'s *typePtr* member to point to the list type's **Tcl_ObjType** structure.

Do not release *objPtr*'s old internal representation unless you replace it with a new one or reset the *typePtr* member to NULL.

The *setFromAnyProc* member may be set to NULL, if the routines making use of the internal representation have no need to derive that internal representation from an arbitrary string value. However, in this case, passing a pointer to the type to Tcl_ConvertToType() will lead to a panic, so to avoid this possibility, the type should *not* be registered.

---

**THE UPDATESTRINGPROC FIELD**

The *updateStringProc* member contains the address of a function called to create a valid string representation from an object's internal representation.
typedef void (Tcl_UpdateStringProc) (Tcl_Obj *objPtr);

`objPtr`'s `bytes` member is always NULL when it is called. It must always set `bytes` non-NULL before returning. We require the string representation's byte array to have a null after the last byte, at offset `length`, and to have no null bytes before that; this allows string representations to be treated as conventional null character-terminated C strings. These restrictions are easily met by using Tcl's internal UTF encoding for the string representation, same as one would do for other Tcl routines accepting string values as arguments. Storage for the byte array must be allocated in the heap by `Tcl_Alloc` or `ckalloc`. Note that `updateStringProcs` must allocate enough storage for the string's bytes and the terminating null byte.

The `updateStringProc` for Tcl's built-in double type, for example, calls `Tcl_PrintDouble` to write to a buffer of size `TCL_DOUBLE_SPACE`, then allocates and copies the string representation to just enough space to hold it. A pointer to the allocated space is stored in the `bytes` member.

The `updateStringProc` member may be set to NULL, if the routines making use of the internal representation are written so that the string representation is never invalidated. Failure to meet this obligation will lead to panics or crashes when `Tcl_GetStringFromObj` or other similar routines ask for the string representation.

**THE DUPINTREPPROC FIELD**

The `dupIntRepProc` member contains the address of a function called to copy an internal representation from one object to another.

typedef void (Tcl_DupInternalRepProc) (Tcl_Obj *srcPtr, Tcl_Obj *dupPtr);
dupPtr's internal representation is made a copy of srcPtr's internal representation. Before the call, srcPtr's internal representation is valid and dupPtr's is not. srcPtr's object type determines what copying its internal representation means.

For example, the dupIntRepProc for the Tcl integer type simply copies an integer. The built-in list type's dupIntRepProc uses a far more sophisticated scheme to continue sharing storage as much as it reasonably can.

THE FREEINTREPPROC FIELD

The freeIntRepProc member contains the address of a function that is called when an object is freed.

```c
typedef void (Tcl_FreeInternalRepProc) (Tcl_Obj *obj);
```

The freeIntRepProc function can deallocate the storage for the object's internal representation and do other type-specific processing necessary when an object is freed.

For example, the list type's freeIntRepProc respects the storage sharing scheme established by the dupIntRepProc so that it only frees storage when the last object sharing it is being freed.

The freeIntRepProc member can be set to NULL to indicate that the internal representation does not require freeing. The freeIntRepProc implementation must not access the bytes member of the object, since Tcl makes its own internal uses of that field during object deletion. The defined tasks for the freeIntRepProc have no need to consult the bytes member.

SEE ALSO

Tcl_NewObj, Tcl_DecrRefCount, Tcl_IncrRefCount
NAME
Tcl_SetObjResult, Tcl_GetObjResult, Tcl_SetResult,
Tcl_GetStringResult, Tcl_AppendResult, Tcl_AppendResultVA,
Tcl_AppendElement, Tcl_ResetResult, Tcl_FreeResult - manipulate Tcl result

SYNOPSIS
#include <tcl.h>
Tcl_SetObjResult(interp, objPtr)
Tcl_Obj *
Tcl_GetObjResult(interp)
Tcl_SetResult(interp, result, freeProc)
const char *
Tcl_GetStringResult(interp)
Tcl_AppendResult(interp, result, result, ... , (char *) NULL)
Tcl_AppendResultVA(interp, argList)
Tcl_AppendElement(interp, element)
Tcl_ResetResult(interp)
Tcl_FreeResult(interp)

ARGUMENTS
DESCRIPTION
OLD STRING PROCEDURES
DIRECT ACCESS TO interp->RESULT IS DEPRECATED
THE Tcl_FREEPROC ARGUMENT TO Tcl_SETRESULT
SEE ALSO
KEYWORDS

NAME
Tcl_SetObjResult, Tcl_GetObjResult, Tcl_SetResult,
Tcl_GetStringResult, Tcl_AppendResult, Tcl_AppendResultVA,
Tcl_AppendElement, Tcl_ResetResult, Tcl_FreeResult - manipulate Tcl
SYNOPSIS

```
#include <tcl.h>
Tcl_SetObjResult(interp, objPtr)
Tcl_Obj *
Tcl_GetObjResult(interp)
Tcl_SetResult(interp, result, freeProc)
const char *
Tcl_GetStringResult(interp)
Tcl_AppendResult(interp, result, result, ..., (char *) NULL)
Tcl_AppendResultVA(interp, argList)
Tcl_AppendElement(interp, element)
Tcl_ResetResult(interp)
Tcl_FreeResult(interp)
```

ARGUMENTS

```
Tcl_Interp  *interp (out)  Interpreter whose result is to be modified or read.
Tcl_Obj *objPtr (in) Object value to become result for interp.
char *result (in) String value to become result for interp or to be appended to the existing result.
char *element (in) String value to append as a list element to the existing result of interp.
Tcl_FreeProc *freeProc (in) Address of procedure to call to release storage at result, or TCL_STATIC.
```
TCL_DYNAMIC, or TCL_VOLATILE.

va_list argList (in)  An argument list which must have been initialized using va_start, and cleared using va_end.

DESCRIPTION

The procedures described here are utilities for manipulating the result value in a Tcl interpreter. The interpreter result may be either a Tcl object or a string. For example, Tcl_SetObjResult and Tcl_SetResult set the interpreter result to, respectively, an object and a string. Similarly, Tcl_GetObjResult and Tcl_GetStringResult return the interpreter result as an object and as a string. The procedures always keep the string and object forms of the interpreter result consistent. For example, if Tcl_SetObjResult is called to set the result to an object, then Tcl_GetStringResult is called, it will return the object's string value.

Tcl_SetObjResult arranges for objPtr to be the result for interp, replacing any existing result. The result is left pointing to the object referenced by objPtr. objPtr's reference count is incremented since there is now a new reference to it from interp. The reference count for any old result object is decremented and the old result object is freed if no references to it remain.

Tcl_GetObjResult returns the result for interp as an object. The object's reference count is not incremented; if the caller needs to retain a long-term pointer to the object they should use Tcl_IncrRefCount to increment its reference count in order to keep it from being freed too early or accidentally changed.

Tcl_SetResult arranges for result to be the result for the current Tcl command in interp, replacing any existing result. The freeProc argument specifies how to manage the storage for the result argument;
it is discussed in the section THE TCL_FREEPROC ARGUMENT TO TCL_SETRESULT below. If result is NULL, then freeProc is ignored and Tcl_SetResult re-initializes interp's result to point to an empty string.

Tcl_GetStringResult returns the result for interp as a string. If the result was set to an object by a Tcl_SetObjResult call, the object form will be converted to a string and returned. If the object's string representation contains null bytes, this conversion will lose information. For this reason, programmers are encouraged to write their code to use the new object API procedures and to call Tcl_GetObjResult instead.

Tcl_ResetResult clears the result for interp and leaves the result in its normal empty initialized state. If the result is an object, its reference count is decremented and the result is left pointing to an unshared object representing an empty string. If the result is a dynamically allocated string, its memory is free'd and the result is left as a empty string. Tcl_ResetResult also clears the error state managed by Tcl_AddErrorInfo, Tcl_AddObjErrorInfo, and Tcl_SetErrorCode.

Tcl_AppendResult makes it easy to build up Tcl results in pieces. It takes each of its result arguments and appends them in order to the current result associated with interp. If the result is in its initialized empty state (e.g. a command procedure was just invoked or Tcl_ResetResult was just called), then Tcl_AppendResult sets the result to the concatenation of its result arguments. Tcl_AppendResult may be called repeatedly as additional pieces of the result are produced. Tcl_AppendResult takes care of all the storage management issues associated with managing interp's result, such as allocating a larger result area if necessary. It also manages conversion to and from the result field of the interp so as to handle backward-compatibility with old-style extensions. Any number of result arguments may be passed in a single call; the last argument in the list must be a NULL pointer.

Tcl_AppendResultVA is the same as Tcl_AppendResult except that instead of taking a variable number of arguments it takes an argument list.
Use of the following procedures (is deprecated since they manipulate the Tcl result as a string. Procedures such as *Tcl_SetObjResult* that manipulate the result as an object can be significantly more efficient.

**Tcl_AppendElement** is similar to **Tcl_AppendResult** in that it allows results to be built up in pieces. However, **Tcl_AppendElement** takes only a single *element* argument and it appends that argument to the current result as a proper Tcl list element. **Tcl_AppendElement** adds backslashes or braces if necessary to ensure that *interp*’s result can be parsed as a list and that *element* will be extracted as a single element. Under normal conditions, **Tcl_AppendElement** will add a space character to *interp*’s result just before adding the new list element, so that the list elements in the result are properly separated. However if the new list element is the first in a list or sub-list (i.e. *interp*’s current result is empty, or consists of the single character “{”, or ends in the characters “ {”) then no space is added.

**Tcl_FreeResult** performs part of the work of **Tcl_ResetResult**. It frees up the memory associated with *interp*’s result. It also sets *interp-* >freeProc to zero, but does not change *interp-*result or clear error state. **Tcl_FreeResult** is most commonly used when a procedure is about to replace one result value with another.

**DIRECT ACCESS TO INTERP-RESULT IS DEPRECATED**

It used to be legal for programs to directly read and write *interp-*result to manipulate the interpreter result. Direct access to *interp-*result is now strongly deprecated because it can make the result’s string and object forms inconsistent. Programs should always read the result using the procedures **Tcl_GetObjResult** or **Tcl_GetStringResult**, and write the result using **Tcl_SetObjResult** or **Tcl_SetResult**.

**THE TCL_FREEPROC ARGUMENT TO TCL_SETRESULT**

**Tcl_SetResult**’s *freeProc* argument specifies how the Tcl system is to manage the storage for the *result* argument. If **Tcl_SetResult** or
**Tcl_SetObjResult** are called at a time when *interp* holds a string result, they do whatever is necessary to dispose of the old string result (see the **Tcl_Interp** manual entry for details on this).

If *freeProc* is **TCL_STATIC** it means that *result* refers to an area of static storage that is guaranteed not to be modified until at least the next call to **Tcl_Eval**. If *freeProc* is **TCL_DYNAMIC** it means that *result* was allocated with a call to **Tcl_Alloc** and is now the property of the Tcl system. **Tcl_SetResult** will arrange for the string's storage to be released by calling **Tcl_Free** when it is no longer needed. If *freeProc* is **TCL_VOLATILE** it means that *result* points to an area of memory that is likely to be overwritten when **Tcl_SetResult** returns (e.g. it points to something in a stack frame). In this case **Tcl_SetResult** will make a copy of the string in dynamically allocated storage and arrange for the copy to be the result for the current Tcl command.

If *freeProc* is not one of the values **TCL_STATIC**, **TCL_DYNAMIC**, and **TCL_VOLATILE**, then it is the address of a procedure that Tcl should call to free the string. This allows applications to use non-standard storage allocators. When Tcl no longer needs the storage for the string, it will call *freeProc*. *FreeProc* should have arguments and result that match the type **Tcl_FreeProc**:

```
typedef void Tcl_FreeProc(char *blockPtr);
```

When *freeProc* is called, its *blockPtr* will be set to the value of *result* passed to **Tcl_SetResult**.

**SEE ALSO**

**Tcl_AddErrorInfo**, **Tcl_CreateObjCommand**, **Tcl_SetErrorCode**, **Tcl_Interp**

**KEYWORDS**

append, command, element, list, object, result, return value, interpreter
NAME
Tcl_AppendExportList, Tcl_CreateNamespace,
Tcl_DeleteNamespace, Tcl_Export, Tcl_FindCommand,
Tcl_FindNamespace, Tcl_ForgetImport,
Tcl_GetCurrentNamespace, Tcl_GetGlobalNamespace,
Tcl_GetNamespaceUnknownHandler, Tcl_Import,
Tcl_SetNamespaceUnknownHandler - manipulate namespaces

SYNOPSIS
#include <tcl.h>
Tcl_Namespace *
Tcl_CreateNamespace(interp, name, clientData, deleteProc)
Tcl_DeleteNamespace(nsPtr)
int
Tcl_AppendExportList(interp, nsPtr, objPtr)
int
Tcl_Export(interp, nsPtr, pattern, resetListFirst)
int
Tcl_Import(interp, nsPtr, pattern, allowOverwrite)
int
Tcl_ForgetImport(interp, nsPtr, pattern)
Tcl_Namespace *
Tcl_GetCurrentNamespace(interp)
Tcl_Namespace *
Tcl_GetGlobalNamespace(interp)
Tcl_Namespace *
Tcl_FindNamespace(interp, name, contextNsPtr, flags)
Tcl_Command
Tcl_FindCommand(interp, name, contextNsPtr, flags)
Tcl_Obj *
Tcl_GetNamespaceUnknownHandler(interp, nsPtr)
int
Tcl_SetNamespaceUnknownHandler(interp, nsPtr, handlerPtr)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS

NAME

Tcl_AppendExportList, Tcl_CreateNamespace, Tcl_DeleteNamespace, Tcl_Export, Tcl_FindCommand, Tcl_FindNamespace, Tcl_ForgetImport, Tcl_GetCurrentNamespace, Tcl_GetGlobalNamespace, Tcl_GetNamespaceUnknownHandler, Tcl_Import, Tcl_SetNamespaceUnknownHandler - manipulate namespaces

SYNOPSIS

#include <tcl.h>
Tcl_Namespace *
Tcl_CreateNamespace(interp, name, clientData, deleteProc)
Tcl_DeleteNamespace(nsPtr)
int
Tcl_AppendExportList(interp, nsPtr, objPtr)
int
Tcl_Export(interp, nsPtr, pattern, resetListFirst)
int
Tcl_Import(interp, nsPtr, pattern, allowOverwrite)
int
Tcl_ForgetImport(interp, nsPtr, pattern)
Tcl_Namespace *
Tcl_GetCurrentNamespace(interp)
Tcl_Namespace *
Tcl_GetGlobalNamespace(interp)
Tcl_Namespace *
Tcl_FindNamespace(interp, name, contextNsPtr, flags)
Tcl_Command
Tcl_FindCommand(interp, name, contextNsPtr, flags)
Tcl_Obj *
### TCL_GetNamespaceUnknownHandler

```c
int
Tcl_GetNamespaceUnknownHandler(interp, nsPtr)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_Interp <em>interp</em></em> (in/out)</td>
<td>The interpreter in which the namespace exists and where name lookups are performed. Also where error result messages are written.</td>
</tr>
<tr>
<td>*<em>const char <em>name</em></em> (in)</td>
<td>The name of the namespace or command to be created or accessed.</td>
</tr>
<tr>
<td><strong>ClientData clientData</strong> (in)</td>
<td>A context pointer by the creator of the namespace. Not interpreted by Tcl at all.</td>
</tr>
<tr>
<td>*<em>Tcl_NamespaceDeleteProc <em>deleteProc</em></em> (in)</td>
<td>A pointer to function to call when the namespace is deleted, or NULL if no such callback is to be performed.</td>
</tr>
<tr>
<td>*<em>Tcl_Namespace <em>nsPtr</em></em> (in)</td>
<td>The namespace to be manipulated, or NULL (for other than <strong>Tcl_DeleteNamespace</strong>) to manipulate the current namespace.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>objPtr</em></em> (out)</td>
<td>A reference to an unshared object to which</td>
</tr>
</tbody>
</table>
The function output will be written.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const char *pattern (in)</td>
<td>The glob-style pattern (see <a href="#">Tcl_StringMatch</a>) that describes the commands to be imported or exported.</td>
</tr>
<tr>
<td>int resetListFirst (in)</td>
<td>Whether the list of export patterns should be reset before adding the current pattern to it.</td>
</tr>
<tr>
<td>int allowOverwrite (in)</td>
<td>Whether new commands created by this import action can overwrite existing commands.</td>
</tr>
<tr>
<td>Tcl_NameSpace *contextNsPtr (in)</td>
<td>The location in the namespace hierarchy where the search for a namespace or command should be conducted relative to when the search term is not rooted at the global namespace. NULL indicates the current namespace.</td>
</tr>
<tr>
<td>int flags (in)</td>
<td>OR-ed combination of bits controlling how the search is to be performed. The following flags are supported: <a href="#">TCL_GLOBAL_ONLY</a> (indicates that the search is always to be conducted</td>
</tr>
</tbody>
</table>
relative to the global namespace),
**TCL_NAMESPACE_ONLY**
(just for
**Tcl_FindCommand**;
indicates that the search is always to be conducted relative to the context namespace), and
**TCL_LEAVE_ERR_MSG**
(indicates that an error message should be left in the interpreter if the search fails.)

| **Tcl_Obj *`handlerPtr`** (in) | A script fragment to be installed as the unknown command handler for the namespace, or NULL to reset the handler to its default. |

---

**DESCRIPTION**

Namespaces are hierarchic naming contexts that can contain commands and variables. They also maintain a list of patterns that describes what commands are exported, and can import commands that have been exported by other namespaces. Namespaces can also be manipulated through the Tcl command **namespace**.

The **Tcl_NameSpace** structure encapsulates a namespace, and is guaranteed to have the following fields in it: **name** (the local name of the namespace, with no namespace separator characters in it, with empty denoting the global namespace), **fullName** (the fully specified name of the namespace), **clientData**, **deleteProc** (the values specified in the call to **Tcl_CreateNamespace**), and **parentPtr** (a pointer to the containing
namespace, or NULL for the global namespace.)

**Tcl_CreateNamespace** creates a new namespace. The *deleteProc* will have the following type signature:

```c
typedef void (Tcl_NamespaceDeleteProc) (ClientData clientData);
```

**Tcl_DeleteNamespace** deletes a namespace.

**Tcl_AppendExportList** retrieves the export patterns for a namespace given namespace and appends them (as list items) to *objPtr*.

**Tcl_Export** sets and appends to the export patterns for a namespace. Patterns are appended unless the *resetListFirst* flag is true.

**Tcl_Import** imports commands matching a pattern into a namespace. Note that the pattern must include the name of the namespace to import from. This function returns an error if an attempt to import a command over an existing command is made, unless the *allowOverwrite* flag has been set.

**Tcl_ForgetImport** removes imports matching a pattern.

**Tcl_GetCurrentNamespace** returns the current namespace for an interpreter.

**Tcl_GetGlobalNamespace** returns the global namespace for an interpreter.

**Tcl_FindNamespace** searches for a namespace named *name* within the context of the namespace *contextNsPtr*. If the namespace cannot be found, NULL is returned.

**Tcl_FindCommand** searches for a command named *name* within the context of the namespace *contextNsPtr*. If the command cannot be found, NULL is returned.
**Tcl_GetNamespaceUnknownHandler** returns the unknown command handler for the namespace, or NULL if none is set.

**Tcl_SetNamespaceUnknownHandler** sets the unknown command handler for the namespace. If *handlerPtr* is NULL, then the handler is reset to its default.

---

**SEE ALSO**

[Tcl_CreateCommand](#), [Tcl_ListObjAppendElements](#), [Tcl_SetVar](#)

---

**KEYWORDS**

namespace, command

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2003 Donal K. Fellows
**NAME**

Tcl_PkgRequire, Tcl_PkgRequireEx, Tcl_PkgRequireProc,
Tcl_PkgPresent, Tcl_PkgPresentEx, Tcl_PkgProvide,
Tcl_PkgProvideEx - package version control

**SYNOPSIS**

```c
#include <tcl.h>

const char *Tcl_PkgRequire(interp, name, version, exact)
const char *Tcl_PkgRequireEx(interp, name, version, exact, clientDataPtr)
int Tcl_PkgRequireProc(interp, name, objc, objv, clientDataPtr)
const char *Tcl_PkgPresent(interp, name, version, exact)
const char *Tcl_PkgPresentEx(interp, name, version, exact, clientDataPtr)
int Tcl_PkgProvide(interp, name, version)
int Tcl_PkgProvideEx(interp, name, version, clientData)
```

**ARGUMENTS**

**DESCRIPTION**

**KEYWORDS**

**NAME**

Tcl_PkgRequire, Tcl_PkgRequireEx, Tcl_PkgRequireProc,
Tcl_PkgPresent, Tcl_PkgPresentEx, Tcl_PkgProvide, Tcl_PkgProvideEx - package version control

**SYNOPSIS**
#include <tcl.h>
const char *
Tcl_PkgRequire(interp, name, version, exact)
const char *
Tcl_PkgRequireEx(interp, name, version, exact, clientDataPtr)
int
Tcl_PkgRequireProc(interp, name, objc, objv, clientDataPtr)
const char *
Tcl_PkgPresent(interp, name, version, exact)
const char *
Tcl_PkgPresentEx(interp, name, version, exact, clientDataPtr)
int
Tcl_PkgProvide(interp, name, version)
int
Tcl_PkgProvideEx(interp, name, version, clientData)

## ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interp *</td>
<td>Interpreter where package is needed or available.</td>
</tr>
<tr>
<td>name</td>
<td>Name of package.</td>
</tr>
<tr>
<td>version *</td>
<td>A version string consisting of one or more decimal numbers separated by dots.</td>
</tr>
<tr>
<td>exact *</td>
<td>Non-zero means that only the particular version specified by version is acceptable. Zero means that newer versions than version are also acceptable as long as they have the same major version number as version.</td>
</tr>
<tr>
<td>ClientData clientData (in)</td>
<td>Arbitrary value to be associated with the package.</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>ClientData *clientDataPtr (out)</td>
<td>Pointer to place to store the value associated with the matching package. It is only changed if the pointer is not NULL and the function completed successfully.</td>
</tr>
<tr>
<td>int objc (in)</td>
<td>Number of requirements.</td>
</tr>
<tr>
<td>Tcl_Obj* objv[] (in)</td>
<td>Array of requirements.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

These procedures provide C-level interfaces to Tcl's package and version management facilities.

**Tcl_PkgRequire** is equivalent to the `package require` command, **Tcl_PkgPresent** is equivalent to the `package present` command, and **Tcl_PkgProvide** is equivalent to the `package provide` command.

See the documentation for the Tcl commands for details on what these procedures do.

If **Tcl_PkgPresent** or **Tcl_PkgRequire** complete successfully they return a pointer to the version string for the version of the package that is provided in the interpreter (which may be different than `version`); if an error occurs they return NULL and leave an error message in the interpreter's result.

**Tcl_PkgProvide** returns **TCL_OK** if it completes successfully; if an error occurs it returns **TCL_ERROR** and leaves an error message in the interpreter's result.
Tcl_PkgProvideEx, Tcl_PkgPresentEx and Tcl_PkgRequireEx allow the setting and retrieving of the client data associated with the package. In all other respects they are equivalent to the matching functions.

Tcl_PkgRequireProc is the form of package require handling multiple requirements. The other forms are present for backward compatibility and translate their invokations to this form.

KEYWORDS

package, present, provide, require, version
NAME
Tcl_Preserve, Tcl_Release, Tcl_EventuallyFree - avoid freeing storage while it is being used

SYNOPSIS
#include <tcl.h>
Tcl_Preserve(clientData)
Tcl_Release(clientData)
Tcl_EventuallyFree(clientData, freeProc)

ARGUMENTS

DESCRIPTION

SEE ALSO

KEYWORDS

NAME

Tcl_Preserve, Tcl_Release, Tcl_EventuallyFree - avoid freeing storage while it is being used

SYNOPSIS

#include <tcl.h>
Tcl_Preserve(clientData)
Tcl_Release(clientData)
Tcl_EventuallyFree(clientData, freeProc)

ARGUMENTS

ClientData clientData (in)  Token describing structure to be freed or reallocated. Usually a pointer to memory for structure.
These three procedures help implement a simple reference count mechanism for managing storage. They are designed to solve a problem having to do with widget deletion, but are also useful in many other situations. When a widget is deleted, its widget record (the structure holding information specific to the widget) must be returned to the storage allocator. However, it is possible that the widget record is in active use by one of the procedures on the stack at the time of the deletion. This can happen, for example, if the command associated with a button widget causes the button to be destroyed: an X event causes an event-handling C procedure in the button to be invoked, which in turn causes the button's associated Tcl command to be executed, which in turn causes the button to be deleted, which in turn causes the button's widget record to be de-allocated. Unfortunately, when the Tcl command returns, the button's event-handling procedure will need to reference the button's widget record. Because of this, the widget record must not be freed as part of the deletion, but must be retained until the event-handling procedure has finished with it. In other situations where the widget is deleted, it may be possible to free the widget record immediately.

*Tcl_Preserve* and *Tcl_Release* implement short-term reference counts for their *clientData* argument. The *clientData* argument identifies an object and usually consists of the address of a structure. The reference counts guarantee that an object will not be freed until each call to *Tcl_Preserve* for the object has been matched by calls to *Tcl_Release*. There may be any number of unmatched *Tcl_Preserve* calls in effect at once.

*Tcl_EventuallyFree* is invoked to free up its *clientData* argument. It checks to see if there are unmatched *Tcl_Preserve* calls for the object. If not, then *Tcl_EventuallyFree* calls *freeProc* immediately. Otherwise *Tcl_EventuallyFree* records the fact that *clientData* needs eventually to
be freed. When all calls to `Tcl_Preserve` have been matched with calls to `Tcl_Release` then `freeProc` will be called by `Tcl_Release` to do the cleanup.

All the work of freeing the object is carried out by `freeProc`. `FreeProc` must have arguments and result that match the type `Tcl_FreeProc`:

```c
typedef void Tcl_FreeProc(char *blockPtr);
```

The `blockPtr` argument to `freeProc` will be the same as the `clientData` argument to `Tcl_EventuallyFree`. The type of `blockPtr (char *)` is different than the type of the `clientData` argument to `Tcl_EventuallyFree` for historical reasons, but the value is the same.

When the `clientData` argument to `Tcl_EventuallyFree` refers to storage allocated and returned by a prior call to `Tcl_Alloc`, `ckalloc`, or another function of the Tcl library, then the `freeProc` argument should be given the special value of `TCL_DYNAMIC`.

This mechanism can be used to solve the problem described above by placing `Tcl_Preserve` and `Tcl_Release` calls around actions that may cause undesired storage re-allocation. The mechanism is intended only for short-term use (i.e. while procedures are pending on the stack); it will not work efficiently as a mechanism for long-term reference counts. The implementation does not depend in any way on the internal structure of the objects being freed; it keeps the reference counts in a separate structure.

SEE ALSO

`Tcl_Interp`, `Tcl_Alloc`

KEYWORDS

free, reference_count, storage

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
Tcl_AppInit - perform application-specific initialization

SYNOPSIS

#include <tcl.h>
int
Tcl_AppInit(interp)

ARGUMENTS

Tcl_Interp *interp (in)  Interpreter for the application.

DESCRIPTION

Tcl_AppInit is a “hook” procedure that is invoked by the main programs for Tcl applications such as tclsh and wish. Its purpose is to allow new Tcl applications to be created without modifying the main programs provided as part of Tcl and Tk. To create a new application you write a
new version of **Tcl_AppInit** to replace the default version provided by Tcl, then link your new **Tcl_AppInit** with the Tcl library.

**Tcl_AppInit** is invoked by **Tcl_Main** and **Tk_Main** after their own initialization and before entering the main loop to process commands. Here are some examples of things that **Tcl_AppInit** might do:

[1] Call initialization procedures for various packages used by the application. Each initialization procedure adds new commands to interp for its package and performs other package-specific initialization.

[2] Process command-line arguments, which can be accessed from the Tcl variables **argv** and **argv0** in interp.

[3] Invoke a startup script to initialize the application.

**Tcl_AppInit** returns **TCL_OK** or **TCL_ERROR**. If it returns **TCL_ERROR** then it must leave an error message in for the interpreter's result; otherwise the result is ignored.

In addition to **Tcl_AppInit**, your application should also contain a procedure **main** that calls **Tcl_Main** as follows:

```c
Tcl_Main(argc, argv, Tcl_AppInit);
```

The third argument to **Tcl_Main** gives the address of the application-specific initialization procedure to invoke. This means that you do not have to use the name **Tcl_AppInit** for the procedure, but in practice the name is nearly always **Tcl_AppInit** (in versions before Tcl 7.4 the name **Tcl_AppInit** was implicit; there was no way to specify the procedure explicitly). The best way to get started is to make a copy of the file **tclAppInit.c** from the Tcl library or source directory. It already contains a **main** procedure and a template for **Tcl_AppInit** that you can modify.
for your application.

**KEYWORDS**

application, argument, command, initialization, interpreter

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tcl_PrintDouble - Convert floating value to string

SYNOPSIS

#include <tcl.h>
Tcl_PrintDouble(interp, value, dst)

ARGUMENTS

**Tcl_Interp** *interp* (in)  
Before Tcl 8.0, the **tcl_precision** variable in this interpreter controlled the conversion. As of Tcl 8.0, this argument is ignored and the conversion is controlled by the **tcl_precision** variable that is now shared by all interpreters.

**double value** (in)  
Floating-point value to be converted.

**char *dst** (out)  
Where to store the string representing *value*. Must have at least **TCL_DOUBLE_SPACE** characters of storage.
**DESCRIPTION**

*Tcl_PrintDouble* generates a string that represents the value of *value* and stores it in memory at the location given by *dst*. It uses `%g` format to generate the string, with one special twist: the string is guaranteed to contain either a “.” or an “e” so that it does not look like an integer. Where `%g` would generate an integer with no decimal point, *Tcl_PrintDouble* adds “.0”.

If the *tcl_precision* value is non-zero, the result will have precisely that many digits of significance. If the value is zero (the default), the result will have the fewest digits needed to represent the number in such a way that *Tcl_NewDoubleObj* will generate the same number when presented with the given string. IEEE semantics of rounding to even apply to the conversion.

**KEYWORDS**

conversion, double-precision, floating-point, string

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME

Tcl_AsyncCreate, Tcl_AsyncMark, Tcl_AsyncInvoke, Tcl_AsyncDelete, Tcl_AsyncReady - handle asynchronous events

SYNOPSIS

#include <tcl.h>
Tcl_AsyncHandler
Tcl_AsyncCreate(proc, clientData)
Tcl_AsyncMark(async)
int
Tcl_AsyncInvoke(interp, code)
Tcl_AsyncDelete(async)
int
Tcl_AsyncReady()
**Tcl AsyncDelete** (*async*)

```c
int Tcl_AsyncDelete(*async)
```

**Tcl AsyncReady**()

### ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl AsyncProc <em>proc</em> (<code>in</code>)</td>
<td>Procedure to invoke to handle an asynchronous event.</td>
</tr>
<tr>
<td>ClientData <em>clientData</em> (<code>in</code>)</td>
<td>One-word value to pass to <code>proc</code>.</td>
</tr>
<tr>
<td>Tcl AsyncHandler <em>async</em> (<code>in</code>)</td>
<td>Token for asynchronous event handler.</td>
</tr>
<tr>
<td>Tcl Interp <em>interp</em> (<code>in</code>)</td>
<td>Tcl interpreter in which command was being evaluated when handler was invoked, or NULL if handler was invoked when there was no interpreter active.</td>
</tr>
<tr>
<td>int <em>code</em> (<code>in</code>)</td>
<td>Completion code from command that just completed in <em>interp</em>, or 0 if <em>interp</em> is NULL.</td>
</tr>
</tbody>
</table>

### DESCRIPTION

These procedures provide a safe mechanism for dealing with asynchronous events such as signals. If an event such as a signal occurs while a Tcl script is being evaluated then it is not safe to take any substantive action to process the event. For example, it is not safe to evaluate a Tcl script since the interpreter may already be in the
middle of evaluating a script; it may not even be safe to allocate memory, since a memory allocation could have been in progress when the event occurred. The only safe approach is to set a flag indicating that the event occurred, then handle the event later when the world has returned to a clean state, such as after the current Tcl command completes.

Tcl_AsyncCreate, Tcl_AsyncDelete, and Tcl_AsyncReady are thread sensitive. They access and/or set a thread-specific data structure in the event of a core built with --enable-threads. The token created by Tcl_AsyncCreate contains the needed thread information it was called from so that calling Tcl_AsyncMark(token) will only yield the origin thread into the asynchronous handler.

Tcl_AsyncCreate creates an asynchronous handler and returns a token for it. The asynchronous handler must be created before any occurrences of the asynchronous event that it is intended to handle (it is not safe to create a handler at the time of an event). When an asynchronous event occurs the code that detects the event (such as a signal handler) should call Tcl_AsyncMark with the token for the handler. Tcl_AsyncMark will mark the handler as ready to execute, but it will not invoke the handler immediately. Tcl will call the proc associated with the handler later, when the world is in a safe state, and proc can then carry out the actions associated with the asynchronous event. Proc should have arguments and result that match the type Tcl_AsyncProc:

```c
typedef int Tcl_AsyncProc(
    ClientData clientData,
    Tcl_Interp *interp,
    int code);
```

The clientData will be the same as the clientData argument passed to Tcl_AsyncCreate when the handler was created. If proc is invoked just after a command has completed execution in an interpreter, then interp will identify the interpreter in which the command was evaluated and
code will be the completion code returned by that command. The command's result will be present in the interpreter's result. When proc returns, whatever it leaves in the interpreter's result will be returned as the result of the command and the integer value returned by proc will be used as the new completion code for the command.

It is also possible for proc to be invoked when no interpreter is active. This can happen, for example, if an asynchronous event occurs while the application is waiting for interactive input or an X event. In this case interp will be NULL and code will be 0, and the return value from proc will be ignored.

The procedure Tcl_AsyncInvoke is called to invoke all of the handlers that are ready. The procedure Tcl_AsyncReady will return non-zero whenever any asynchronous handlers are ready; it can be checked to avoid calls to Tcl_AsyncInvoke when there are no ready handlers. Tcl calls Tcl_AsyncReady after each command is evaluated and calls Tcl_AsyncInvoke if needed. Applications may also call Tcl_AsyncInvoke at interesting times for that application. For example, Tcl's event handler calls Tcl_AsyncReady after each event and calls Tcl_AsyncInvoke if needed. The interp and code arguments to Tcl_AsyncInvoke have the same meaning as for proc: they identify the active interpreter, if any, and the completion code from the command that just completed.

Tcl_AsyncDelete removes an asynchronous handler so that its proc will never be invoked again. A handler can be deleted even when ready, and it will still not be invoked.

If multiple handlers become active at the same time, the handlers are invoked in the order they were created (oldest handler first). The code and the interpreter's result for later handlers reflect the values returned by earlier handlers, so that the most recently created handler has last say about the interpreter's result and completion code. If new handlers become ready while handlers are executing, Tcl_AsyncInvoke will invoke them all; at each point it invokes the highest-priority (oldest) ready handler, repeating this over and over until there are no longer any ready handlers.
It is almost always a bad idea for an asynchronous event handler to modify the interpreter's result or return a code different from its `code` argument. This sort of behavior can disrupt the execution of scripts in subtle ways and result in bugs that are extremely difficult to track down. If an asynchronous event handler needs to evaluate Tcl scripts then it should first save the interpreter's state by calling `Tcl_SaveInterpState`, passing in the `code` argument. When the asynchronous handler is finished it should restore the interpreter's state by calling `Tcl_RestoreInterpState`, and then returning the `code` argument.

**KEYWORDS**

asynchronous event, handler, signal, Tcl_SaveInterpState, thread
NAME

Tcl_PutEnv - procedures to manipulate the environment

SYNOPSIS

#include <tcl.h>
int
Tcl_PutEnv(assignment)

ARGUMENTS

const char *assignment (in) Info about environment variable in the format NAME=value. The assignment argument is in the system encoding.

DESCRIPTION

Tcl_PutEnv sets an environment variable. The information is passed in a single string of the form NAME=value. This procedure is intended to be a stand-in for the UNIX putenv system call. All Tcl-based applications using putenv should redefine it to Tcl_PutEnv so that they will interface properly to the Tcl runtime.

KEYWORDS

environment, variable

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
Tcl_GetTime, Tcl_SetTimeProc, Tcl_QueryTimeProc - get date and time

SYNOPSIS
#include <tcl.h>
Tcl_GetTime(timePtr)
Tcl_SetTimeProc(getProc, scaleProc, clientData)
Tcl_QueryTimeProc(getProcPtr, scaleProcPtr, clientDataPtr)

ARGUMENTS
Tcl_Time * timePtr (out)  Points to memory in which to store the date and time information.
Tcl_GetTimeProc * getProc (in)  Pointer to handler function replacing Tcl_GetTime's
access to the OS.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_ScaleTimeProc * <code>scaleProc</code> (in)</td>
<td>Pointer to handler function for the conversion of time delays in the virtual domain to real-time.</td>
</tr>
<tr>
<td>ClientData * <code>clientData</code> (in)</td>
<td>Value passed through to the two handler functions.</td>
</tr>
<tr>
<td>Tcl_GetTimeProc ** <code>getProcPtr</code> (inout)</td>
<td>Pointer to place the currently registered get handler function into.</td>
</tr>
<tr>
<td>Tcl_ScaleTimeProc ** <code>scaleProcPtr</code> (inout)</td>
<td>Pointer to place the currently registered scale handler function into.</td>
</tr>
<tr>
<td>ClientData ** <code>clientDataPtr</code> (inout)</td>
<td>Pointer to place the currently registered pass-through value into.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

The **Tcl_GetTime** function retrieves the current time as a **Tcl_Time** structure in memory the caller provides. This structure has the following definition:

```c
typedef struct Tcl_Time {
    long sec;
    long usec;
} Tcl_Time;
```

On return, the `sec` member of the structure is filled in with the number of seconds that have elapsed since the *epoch*: the epoch is the point in
time of 00:00 UTC, 1 January 1970. This number does not count leap seconds - an interval of one day advances it by 86400 seconds regardless of whether a leap second has been inserted.

The *usec* member of the structure is filled in with the number of microseconds that have elapsed since the start of the second designated by *sec*. The Tcl library makes every effort to keep this number as precise as possible, subject to the limitations of the computer system. On multiprocessor variants of Windows, this number may be limited to the 10- or 20-ms granularity of the system clock. (On single-processor Windows systems, the *usec* field is derived from a performance counter and is highly precise.)

The **Tcl_SetTime** function registers two related handler functions with the core. The first handler function is a replacement for **Tcl_GetTime**, or rather the OS access made by **Tcl_GetTime**. The other handler function is used by the Tcl notifier to convert wait/block times from the virtual domain into real time.

The **Tcl_QueryTime** function returns the currently registered handler functions. If no external handlers were set then this will return the standard handlers accessing and processing the native time of the OS. The arguments to the function are allowed to be NULL; and any argument which is NULL is ignored and not set.

Any handler pair specified has to return data which is consistent between them. In other words, setting one handler of the pair to something assuming a 10-times slowdown, and the other handler of the pair to something assuming a two-times slowdown is wrong and not allowed.

The set handler functions are allowed to run the delivered time backwards, however this should be avoided. We have to allow it as the native time can run backwards as the user can fiddle with the system time one way or other. Note that the insertion of the hooks will not change the behaviour of the Tcl core with regard to this situation, i.e. the existing behaviour is retained.
SEE ALSO

clock

KEYWORDS

date, time

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2001 by Kevin B. Kenny <kennykb(at)acm.org>.
NAME
Tcl_IsSafe, Tcl_MakeSafe, Tcl_CreateSlave, Tcl_GetSlave,
Tcl_GetMaster, Tcl_GetInterpPath, Tcl_CreateAlias,
Tcl_CreateAliasObj, Tcl_GetAlias, Tcl_GetAliasObj,
Tcl_ExposeCommand, Tcl_HideCommand - manage multiple
Tcl interpreters, aliases and hidden commands

SYNOPSIS
#include <tcl.h>
int
Tcl_IsSafe(interp)
int
Tcl_MakeSafe(interp)
Tcl_Interp *
Tcl_CreateSlave(interp, slaveName, isSafe)
Tcl_Interp *
Tcl_GetSlave(interp, slaveName)
Tcl_Interp *
Tcl_GetMaster(interp)
int
Tcl_GetInterpPath(askingInterp, slaveInterp)
int
Tcl_CreateAlias(slaveInterp, slaveCmd, targetInterp,
targetCmd, argc, argv)
int
Tcl_CreateAliasObj(slaveInterp, slaveCmd, targetInterp,
targetCmd, objc, objv)
int
Tcl_GetAlias(interp, slaveCmd, targetInterpPtr, targetCmdPtr,
argcPtr, argvPtr)
int
Tcl_GetAliasObj(interp, slaveCmd, targetInterpPtr, targetCmdPtr, objcPtr, objvPtr)
int
Tcl_ExposeCommand(interp, hiddenCmdName, cmdName)
int
Tcl_HideCommand(interp, cmdName, hiddenCmdName)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS

NAME

Tcl_IsSafe, Tcl_MakeSafe, Tcl_CreateSlave, Tcl_GetSlave, Tcl_GetMaster, Tcl_GetInterpPath, Tcl_CreateAlias, Tcl_CreateAliasObj, Tcl_GetAlias, Tcl_GetAliasObj, Tcl_ExposeCommand, Tcl_HideCommand - manage multiple Tcl interpreters, aliases and hidden commands

SYNOPSIS

#include <tcl.h>
int
Tcl_IsSafe(interp)
int
Tcl_MakeSafe(interp)
Tcl_Interp *
Tcl_CreateSlave(interp, slaveName, isSafe)
Tcl_Interp *
Tcl_GetSlave(interp, slaveName)
Tcl_Interp *
Tcl_GetMaster(interp)
int
Tcl_GetInterpPath(askingInterp, slaveInterp)
int
Tcl_CreateAlias(slaveInterp, slaveCmd, targetInterp, targetCmd,
int Tcl_CreateAliasObj(
    slaveInterp, slaveCmd, targetInterp, targetCmd,
    objc, objv
)

int Tcl_GetAlias(
    interp, slaveCmd, targetInterpPtr, targetCmdPtr,
    argcPtr, argvPtr
)

int Tcl_GetAliasObj(
    interp, slaveCmd, targetInterpPtr, targetCmdPtr,
    objcPtr, objvPtr
)

int Tcl_ExposeCommand(
    interp, hiddenCmdName, cmdName
)

int Tcl_HideCommand(
    interp, cmdName, hiddenCmdName
)

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_Interp <em>interp</em></em> (in)</td>
<td>Interpreter in which to execute the specified command.</td>
</tr>
<tr>
<td>const char *slaveName (in)</td>
<td>Name of slave interpreter to create or manipulate.</td>
</tr>
<tr>
<td>int isSafe (in)</td>
<td>If non-zero, a “safe” slave that is suitable for running untrusted code is created, otherwise a trusted slave is created.</td>
</tr>
<tr>
<td>*<em>Tcl_Interp <em>slaveInterp</em></em> (in)</td>
<td>Interpreter to use for creating the source command for an alias (see below).</td>
</tr>
<tr>
<td>const char *slaveCmd (in)</td>
<td>Name of source command for alias.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>Tcl_Interp</strong> <em>targetInterp</em> (in)</td>
<td>Interpreter that contains the target command for an alias.</td>
</tr>
<tr>
<td>const char <em>targetCmd</em> (in)</td>
<td>Name of target command for alias in <em>targetInterp</em>.</td>
</tr>
<tr>
<td>int argc (in)</td>
<td>Count of additional arguments to pass to the alias command.</td>
</tr>
<tr>
<td>const char *const <em>argv</em> (in)</td>
<td>Vector of strings, the additional arguments to pass to the alias command. This storage is owned by the caller.</td>
</tr>
<tr>
<td>int objc (in)</td>
<td>Count of additional object arguments to pass to the alias object command.</td>
</tr>
<tr>
<td><strong>Tcl_Obj</strong> <strong>objv</strong> (in)</td>
<td>Vector of Tcl_Obj structures, the additional object arguments to pass to the alias object command. This storage is owned by the caller.</td>
</tr>
<tr>
<td><strong>Tcl_Interp</strong> <strong>targetInterpPtr</strong> (in)</td>
<td>Pointer to location to store the address of the interpreter where a target command is defined for an alias.</td>
</tr>
<tr>
<td>const char <strong>targetCmdPtr</strong> (out)</td>
<td>Pointer to location to store the address of the name of the target command for an alias.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>int *argcPtr (out)</td>
<td>Pointer to location to store count of additional arguments to be passed to the alias. The location is in storage owned by the caller.</td>
</tr>
<tr>
<td>const char ***argvPtr (out)</td>
<td>Pointer to location to store a vector of strings, the additional arguments to pass to an alias. The location is in storage owned by the caller, the vector of strings is owned by the called function.</td>
</tr>
<tr>
<td>int *objcPtr (out)</td>
<td>Pointer to location to store count of additional object arguments to be passed to the alias. The location is in storage owned by the caller.</td>
</tr>
<tr>
<td>Tcl_Obj ***objvPtr (out)</td>
<td>Pointer to location to store a vector of Tcl_Obj structures, the additional arguments to pass to an object alias command. The location is in storage owned by the caller, the vector of Tcl_Obj structures is owned by the called function.</td>
</tr>
<tr>
<td>const char *cmdName (in)</td>
<td>Name of an exposed alias.</td>
</tr>
</tbody>
</table>
These procedures are intended for access to the multiple interpreter facility from inside C programs. They enable managing multiple interpreters in a hierarchical relationship, and the management of aliases, commands that when invoked in one interpreter execute a command in another interpreter. The return value for those procedures that return an int is either TCL_OK or TCL_ERROR. If TCL_ERROR is returned then the result field of the interpreter contains an error message.

**Tcl_CreateSlave** creates a new interpreter as a slave of *interp*. It also creates a slave command named *slaveName* in *interp* which allows *interp* to manipulate the new slave. If *isSafe* is zero, the command creates a trusted slave in which Tcl code has access to all the Tcl commands. If it is 1, the command creates a “safe” slave in which Tcl code has access only to set of Tcl commands defined as “Safe Tcl”; see the manual entry for the Tcl *interp* command for details. If the creation of the new slave interpreter failed, NULL is returned.

**Tcl_IsSafe** returns 1 if *interp* is “safe” (was created with the TCL_SAFE_INTERPRETER flag specified), 0 otherwise.

**Tcl_MakeSafe** marks *interp* as “safe”, so that future calls to Tcl_IsSafe will return 1. It also removes all known potentially-unsafe core functionality (both commands and variables) from *interp*. However, it cannot know what parts of an extension or application are safe and does not make any attempt to remove those parts, so safety is not guaranteed after calling Tcl_MakeSafe. Callers will want to take care
with their use of **Tcl_MakeSafe** to avoid false claims of safety. For many situations, **Tcl_CreateSlave** may be a better choice, since it creates interpreters in a known-safe state.

**Tcl_GetSlave** returns a pointer to a slave interpreter of *interp*. The slave interpreter is identified by *slaveName*. If no such slave interpreter exists, **NULL** is returned.

**Tcl_GetMaster** returns a pointer to the master interpreter of *interp*. If *interp* has no master (it is a top-level interpreter) then **NULL** is returned.

**Tcl_GetInterpPath** sets the *result* field in *askingInterp* to the relative path between *askingInterp* and *slaveInterp*; *slaveInterp* must be a slave of *askingInterp*. If the computation of the relative path succeeds, **TCL_OK** is returned, else **TCL_ERROR** is returned and the *result* field in *askingInterp* contains the error message.

**Tcl_CreateAlias** creates an object command named *slaveCmd* in *slaveInterp* that when invoked, will cause the command *targetCmd* to be invoked in *targetInterp*. The arguments specified by the strings contained in *argv* are always prepended to any arguments supplied in the invocation of *slaveCmd* and passed to *targetCmd*. This operation returns **TCL_OK** if it succeeds, or **TCL_ERROR** if it fails; in that case, an error message is left in the object result of *slaveInterp*. Note that there are no restrictions on the ancestry relationship (as created by **Tcl_CreateSlave**) between *slaveInterp* and *targetInterp*. Any two interpreters can be used, without any restrictions on how they are related.

**Tcl_CreateAliasObj** is similar to **Tcl_CreateAlias** except that it takes a vector of objects to pass as additional arguments instead of a vector of strings.

**Tcl_GetAlias** returns information about an alias *aliasName* in *interp*. Any of the result fields can be **NULL**, in which case the corresponding datum is not returned. If a result field is non-**NULL**, the address indicated is set to the corresponding datum. For example, if *targetNamePtr* is non-**NULL** it is set to a pointer to the string containing
the name of the target command.

**Tcl_GetAliasObj** is similar to **Tcl_GetAlias** except that it returns a pointer to a vector of Tcl_Obj structures instead of a vector of strings.

**Tcl_ExposeCommand** moves the command named *hiddenCmdName* from the set of hidden commands to the set of exposed commands, putting it under the name *cmdName*. *HiddenCmdName* must be the name of an existing hidden command, or the operation will return **TCL_ERROR** and leave an error message in the *result* field in *interp*. If an exposed command named *cmdName* already exists, the operation returns **TCL_ERROR** and leaves an error message in the object result of *interp*. If the operation succeeds, it returns **TCL_OK**. After executing this command, attempts to use *cmdName* in a call to **Tcl_Eval** or with the Tcl **eval** command will again succeed.

**Tcl_HideCommand** moves the command named *cmdName* from the set of exposed commands to the set of hidden commands, under the name *hiddenCmdName*. *CmdName* must be the name of an existing exposed command, or the operation will return **TCL_ERROR** and leave an error message in the object result of *interp*. Currently both *cmdName* and *hiddenCmdName* must not contain namespace qualifiers, or the operation will return **TCL_ERROR** and leave an error message in the object result of *interp*. The *CmdName* will be looked up in the global namespace, and not relative to the current namespace, even if the current namespace is not the global one. If a hidden command whose name is *hiddenCmdName* already exists, the operation also returns **TCL_ERROR** and the *result* field in *interp* contains an error message. If the operation succeeds, it returns **TCL_OK**. After executing this command, attempts to use *cmdName* in a call to **Tcl_Eval** or with the Tcl **eval** command will fail.

For a description of the Tcl interface to multiple interpreters, see *interp(n)*.

### SEE ALSO

**interp**
KEYWORDS

alias, command, exposed commands, hidden commands, interpreter, invoke, master, slave

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME

Tcl_GetAssocData, Tcl_SetAssocData, Tcl_DeleteAssocData - manage associations of string keys and user specified data with Tcl interpreters

SYNOPSIS

#include <tcl.h>
ClientData
Tcl_GetAssocData(interp, key, delProcPtr)
Tcl_SetAssocData(interp, key, delProc, clientData)
Tcl_DeleteAssocData(interp, key)

ARGUMENTS

DESCRIPTION

KEYWORDS

NAME

Tcl_GetAssocData, Tcl_SetAssocData, Tcl_DeleteAssocData - manage associations of string keys and user specified data with Tcl interpreters

SYNOPSIS

#include <tcl.h>
ClientData
Tcl_GetAssocData(interp, key, delProcPtr)
Tcl_SetAssocData(interp, key, delProc, clientData)
Tcl_DeleteAssocData(interp, key)

ARGUMENTS

Tcl_Interp *interp (in)  Interpreter in which to execute the specified command.
<table>
<thead>
<tr>
<th>*<em>const char <em>key</em></em> (in)</th>
<th>Key for association with which to store data or from which to delete or retrieve data. Typically the module prefix for a package.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_InterpDeleteProc <em>delProc</em></em> (in)</td>
<td>Procedure to call when <em>interp</em> is deleted.</td>
</tr>
<tr>
<td>**Tcl_InterpDeleteProc <strong>delProcPtr</strong> (in)</td>
<td>Pointer to location in which to store address of current deletion procedure for association. Ignored if NULL.</td>
</tr>
<tr>
<td><strong>ClientData clientData</strong> (in)</td>
<td>Arbitrary one-word value associated with the given key in this interpreter. This data is owned by the caller.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

These procedures allow extensions to associate their own data with a Tcl interpreter. An association consists of a string key, typically the name of the extension, and a one-word value, which is typically a pointer to a data structure holding data specific to the extension. Tcl makes no interpretation of either the key or the value for an association.

Storage management is facilitated by storing with each association a procedure to call when the interpreter is deleted. This procedure can dispose of the storage occupied by the client's data in any way it sees fit.

**Tcl_SetAssocData** creates an association between a string key and a user specified datum in the given interpreter. If there is already an
association with the given key, **Tcl_SetAssocData** overwrites it with the new information. It is up to callers to organize their use of names to avoid conflicts, for example, by using package names as the keys. If the deleteProc argument is non-NULL it specifies the address of a procedure to invoke if the interpreter is deleted before the association is deleted. **DeleteProc** should have arguments and result that match the type **Tcl_InterpDeleteProc**:

```c
typedef void Tcl_InterpDeleteProc(
    ClientData clientData,
    Tcl_Interp *interp);
```

When deleteProc is invoked the clientData and interp arguments will be the same as the corresponding arguments passed to **Tcl_SetAssocData**. The deletion procedure will *not* be invoked if the association is deleted before the interpreter is deleted.

**Tcl_GetAssocData** returns the datum stored in the association with the specified key in the given interpreter, and if the delProcPtr field is non-NULL, the address indicated by it gets the address of the delete procedure stored with this association. If no association with the specified key exists in the given interpreter **Tcl_GetAssocData** returns NULL.

**Tcl_DeleteAssocData** deletes an association with a specified key in the given interpreter. Then it calls the deletion procedure.

**KEYWORDS**

association, data, deletion procedure, interpreter, key

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME
Tcl_DetachPids, Tcl_ReapDetachedProcs, Tcl_WaitPid - manage child processes in background

SYNOPSIS
#include <tcl.h>
Tcl_DetachPids(numPids, pidPtr)
Tcl_ReapDetachedProcs()
Tcl_Pid
Tcl_WaitPid(pid, statusPtr, options)

ARGUMENTS
int numPids (in) Number of process ids contained in the array pointed to by pidPtr.
### DESCRIPTION

**Tcl_DetachPids** and **Tcl_ReapDetachedProcs** provide a mechanism for managing subprocesses that are running in background. These procedures are needed because the parent of a process must eventually invoke the **waitpid** kernel call (or one of a few other similar kernel calls) to wait for the child to exit. Until the parent waits for the child, the child's state cannot be completely reclaimed by the system. If a parent continually creates children and doesn't wait on them, the system's process table will eventually overflow, even if all the children have exited.

**Tcl_DetachPids** may be called to ask Tcl to take responsibility for one or more processes whose process ids are contained in the **pidPtr** array passed as argument. The caller presumably has started these processes running in background and does not want to have to deal with them again.

**Tcl_ReapDetachedProcs** invokes the **waitpid** kernel call on each of the background processes so that its state can be cleaned up if it has

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *pidPtr (in)</td>
<td>Address of array containing <strong>numPids</strong> process ids.</td>
</tr>
<tr>
<td>Tcl_Pid <strong>pid</strong> (in)</td>
<td>The id of the process (pipe) to wait for.</td>
</tr>
<tr>
<td>int *statusPtr (out)</td>
<td>The result of waiting on a process (pipe). Either 0 or ECHILD.</td>
</tr>
<tr>
<td>int <strong>options</strong> (in)</td>
<td>The options controlling the wait. WNOHANG specifies not to wait when checking the process.</td>
</tr>
</tbody>
</table>
exited. If the process has not exited yet, `Tcl_ReapDetachedProcs` does not wait for it to exit; it will check again the next time it is invoked. Tcl automatically calls `Tcl_ReapDetachedProcs` each time the `exec` command is executed, so in most cases it is not necessary for any code outside of Tcl to invoke `Tcl_ReapDetachedProcs`. However, if you call `Tcl_DetachPids` in situations where the `exec` command may never get executed, you may wish to call `Tcl_ReapDetachedProcs` from time to time so that background processes can be cleaned up.

`Tcl_WaitPid` is a thin wrapper around the facilities provided by the operating system to wait on the end of a spawned process and to check a whether spawned process is still running. It is used by `Tcl_ReapDetachedProcs` and the channel system to portably access the operating system.

**KEYWORDS**

`background`, `child`, `detach`, `process`, `wait`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tcl_BackgroundError - report Tcl error that occurred in background processing

SYNOPSIS

#include <tcl.h>
Tcl_BackgroundError(interp)

ARGUMENTS

Tcl_Interp *interp (in)  Interpreter in which the error occurred.

DESCRIPTION

This procedure is typically invoked when a Tcl error occurs during "background processing" such as executing an event handler. When such an error occurs, the error condition is reported to Tcl or to a widget or some other C code, and there is not usually any obvious way for that code to report the error to the user. In these cases the code calls Tcl_BackgroundError with an interp argument identifying the interpreter in which the error occurred. At the time Tcl_BackgroundError is invoked, the interpreter's result is expected to contain an error message. Tcl_BackgroundError will invoke the command registered in that interpreter to handle background errors by the interp bgerror command. The registered handler command is meant to report the error in an application-specific fashion. The handler command receives two arguments, the result of the interp, and the return options of the interp at the time the error occurred. If the
application registers no handler command, the default handler command will attempt to call `bgerror` to report the error. If an error condition arises while invoking the handler command, then `Tcl_BackgroundError` reports the error itself by printing a message on the standard error file.

`Tcl_BackgroundError` does not invoke the handler command immediately because this could potentially interfere with scripts that are in process at the time the error occurred. Instead, it invokes the handler command later as an idle callback.

It is possible for many background errors to accumulate before the handler command is invoked. When this happens, each of the errors is processed in order. However, if the handle command returns a break exception, then all remaining error reports for the interpreter are skipped.

**KEYWORDS**

- `background`
- `bgerror`
- `error`
- ` interp`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1992-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tcl_NewIntObj, Tcl_NewLongObj, Tcl_NewWideIntObj,
Tcl_SetIntObj, Tcl_SetLongObj, Tcl_SetWideIntObj,
Tcl_GetIntFromObj, Tcl_GetLongFromObj,
Tcl_GetWideIntFromObj, Tcl_NewBignumObj,
Tcl_SetBignumObj, Tcl_GetBignumFromObj,
Tcl_TakeBignumFromObj - manipulate Tcl objects as integer values

SYNOPSIS
#include <tcl.h>
Tcl_Obj *
Tcl_NewIntObj(intValue)
Tcl_Obj *
Tcl_NewLongObj(longValue)
Tcl_Obj *
Tcl_NewWideIntObj(wideValue)
Tcl_SetIntObj(objPtr, intValue)
Tcl_SetLongObj(objPtr, longValue)
Tcl_SetWideIntObj(objPtr, wideValue)
int
Tcl_GetIntFromObj(interp, objPtr, intptr)
int
Tcl_GetLongFromObj(interp, objPtr, longPtr)
int
Tcl_GetWideIntFromObj(interp, objPtr, widePtr)
#include <tclTomMath.h>
Tcl_Obj *
Tcl_NewBignumObj(bigValue)
Tcl_SetBignumObj(objPtr, bigValue)
int
Tcl_GetBignumFromObj(interp, objPtr, bigValue)
int
Tcl_TakeBignumFromObj(interp, objPtr, bigValue)
int
Tcl_InitBignumFromDouble(interp, doubleValue, bigValue)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS

NAME
Tcl_NewIntObj, Tcl_NewLongObj, Tcl_NewWideIntObj, Tcl_SetIntObj,
Tcl_SetLongObj, Tcl_SetWideIntObj, Tcl_GetIntFromObj,
Tcl_GetLongFromObj, Tcl_GetWideIntFromObj, Tcl_NewBignumObj,
Tcl_SetBignumObj, Tcl_GetBignumFromObj, Tcl_TakeBignumFromObj -
manipulate Tcl objects as integer values

SYNOPSIS
#include <tcl.h>
Tcl_Obj *
Tcl_NewIntObj(intValue)
Tcl_Obj *
Tcl_NewLongObj(longValue)
Tcl_Obj *
Tcl_NewWideIntObj(wideValue)
Tcl_SetIntObj(objPtr, intValue)
Tcl_SetLongObj(objPtr, longValue)
Tcl_SetWideIntObj(objPtr, wideValue)
int
Tcl_GetIntFromObj(interp, objPtr, intPtr)
int
Tcl_GetLongFromObj(interp, objPtr, longPtr)
int
Tcl_GetWideIntFromObj(interp, objPtr, widePtr)
#include <tclTomMath.h>
Tcl_Obj *
Tcl_NewBignumObj(bigValue)
**Tcl_SetBignumObj**(objPtr, bigValue)

int

**Tcl_GetBignumFromObj**(interp, objPtr, bigValue)

int

**Tcl_TakeBignumFromObj**(interp, objPtr, bigValue)

int

**Tcl_InitBignumFromDouble**(interp, doubleValue, bigValue)

---

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int intValue (in)</td>
<td>Integer value used to initialize or set a Tcl object.</td>
</tr>
<tr>
<td>long longValue (in)</td>
<td>Long integer value used to initialize or set a Tcl object.</td>
</tr>
<tr>
<td>Tcl_WideInt wideValue (in)</td>
<td>Wide integer value used to initialize or set a Tcl object.</td>
</tr>
<tr>
<td>Tcl_Obj *objPtr (in/out)</td>
<td>For Tcl_SetIntObj, Tcl_SetLongObj, Tcl_SetWideIntObj, and Tcl_SetBignumObj, this points to the object in which to store an integral value. For Tcl_GetIntFromObj, Tcl_GetLongFromObj, Tcl_GetWideIntFromObj, Tcl_GetBignumFromObj, and Tcl_TakeBignumFromObj, this refers to the object from which to retrieve an integral value.</td>
</tr>
<tr>
<td>Tcl_Interp *interp (in/out)</td>
<td>When non-NULL, an error</td>
</tr>
</tbody>
</table>
message is left here when integral value retrieval fails.

<table>
<thead>
<tr>
<th>Parameter Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *intPtr (out)</td>
<td>Points to place to store the integer value retrieved from objPtr.</td>
</tr>
<tr>
<td>long *longPtr (out)</td>
<td>Points to place to store the long integer value retrieved from objPtr.</td>
</tr>
<tr>
<td>Tcl_WideInt *widePtr (out)</td>
<td>Points to place to store the wide integer value retrieved from objPtr.</td>
</tr>
<tr>
<td>mp_int *bigValue (in/out)</td>
<td>Points to a multi-precision integer structure declared by the LibTomMath library.</td>
</tr>
<tr>
<td>double doubleValue (in)</td>
<td>Double value from which the integer part is determined and used to initialize a multi-precision integer value.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

These procedures are used to create, modify, and read Tcl objects that hold integral values.

The different routines exist to accommodate different integral types in C with which values might be exchanged. The C integral types for which Tcl provides value exchange routines are **int**, **long int**, **Tcl_WideInt**, and **mp_int**. The **int** and **long int** types are provided by the C language standard. The **Tcl_WideInt** type is a typedef defined to be whatever
signed integral type covers at least the 64-bit integer range (-9223372036854775808 to 9223372036854775807). Depending on the platform and the C compiler, the actual type might be `long int`, `long long int`, `int64`, or something else. The `mp_int` type is a multiple-precision integer type defined by the LibTomMath multiple-precision integer library.

The `Tcl_NewIntObj`, `Tcl_NewLongObj`, `Tcl_NewWideIntObj`, and `Tcl_NewBignumObj` routines each create and return a new Tcl object initialized to the integral value of the argument. The returned Tcl object is unshared.

The `Tcl_SetIntObj`, `Tcl_SetLongObj`, `Tcl_SetWideIntObj`, and `Tcl_SetBignumObj` routines each set the value of an existing Tcl object pointed to by `objPtr` to the integral value provided by the other argument. The `objPtr` argument must point to an unshared Tcl object. Any attempt to set the value of a shared Tcl object violates Tcl's copy-on-write policy. Any existing string representation or internal representation in the unshared Tcl object will be freed as a consequence of setting the new value.

The `Tcl_GetIntFromObj`, `Tcl_GetLongFromObj`, `Tcl_GetWideIntFromObj`, `Tcl_GetBignumFromObj`, and `Tcl_TakeBignumFromObj` routines attempt to retrieve an integral value of the appropriate type from the Tcl object `objPtr`. If the attempt succeeds, then `TCL_OK` is returned, and the value is written to the storage provided by the caller. The attempt might fail if `objPtr` does not hold an integral value, or if the value exceeds the range of the target type. If the attempt fails, then `TCL_ERROR` is returned, and if `interp` is non-NULL, an error message is left in `interp`. The `Tcl_ObjType` of `objPtr` may be changed to make subsequent calls to the same routine more efficient. Unlike the other functions, `Tcl_TakeBignumFromObj` may set the content of the Tcl object `objPtr` to an empty string in the process of retrieving the multiple-precision integer value.

The choice between `Tcl_GetBignumFromObj` and `Tcl_TakeBignumFromObj` is governed by how the caller will continue to use `objPtr`. If after the `mp_int` value is retrieved from `objPtr`, the
caller will make no more use of \textit{objPtr}, then using \texttt{Tcl\_TakeBignumFromObj} permits Tcl to detect when an unshared \textit{objPtr} permits the value to be moved instead of copied, which should be more efficient. If anything later in the caller requires \textit{objPtr} to continue to hold the same value, then \texttt{Tcl\_GetBignumFromObj} must be chosen.

The \texttt{Tcl\_InitBignumFromDouble} routine is a utility procedure that extracts the integer part of \textit{doubleValue} and stores that integer value in the \texttt{mp\_int} value \textit{bigValue}.

\textbf{SEE ALSO}

\texttt{Tcl\_NewObj}, \texttt{Tcl\_DecrRefCount}, \texttt{Tcl\_IncrRefCount}, \texttt{Tcl\_GetObjResult}

\textbf{KEYWORDS}

\texttt{integer, integer object, integer type, internal representation, object, object type, string representation}
NAME

Tcl_RecordAndEval - save command on history list before evaluating

SYNOPSIS

#include <tcl.h>

int
Tcl_RecordAndEval(interp, cmd, flags)

ARGUMENTS

Tcl_Interp *interp (in)  Tcl interpreter in which to evaluate command.

const char *cmd (in)  Command (or sequence of commands) to execute.

int flags (in)  An OR'ed combination of flag bits. TCL_NO_EVAL means record the command but do not evaluate it. TCL_EVAL_GLOBAL means evaluate the command at global level instead of the current stack level.

DESCRIPTION
Tcl_RecordAndEval is invoked to record a command as an event on the history list and then execute it using Tcl_Eval (or Tcl_GlobalEval if the TCL_EVAL_GLOBAL bit is set in flags). It returns a completion code such as TCL_OK just like Tcl_Eval and it leaves information in the interpreter's result. If you do not want the command recorded on the history list then you should invoke Tcl_Eval instead of Tcl_RecordAndEval. Normally Tcl_RecordAndEval is only called with top-level commands typed by the user, since the purpose of history is to allow the user to re-issue recently-invoked commands. If the flags argument contains the TCL_NO_EVAL bit then the command is recorded without being evaluated.

Note that Tcl_RecordAndEval has been largely replaced by the object-based procedure Tcl_RecordAndEvalObj. That object-based procedure records and optionally executes a command held in a Tcl object instead of a string.

SEE ALSO

Tcl_RecordAndEvalObj

KEYWORDS

command, event, execute, history, interpreter, record

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME

Tcl_Backslash - parse a backslash sequence

SYNOPSIS

```c
#include <tcl.h>
char
Tcl_Backslash(src, countPtr)
```

ARGUMENTS

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *src</td>
<td>Pointer to a string starting with a backslash.</td>
</tr>
<tr>
<td>int *countPtr</td>
<td>If countPtr is not NULL, *countPtr gets filled in with number of characters in the backslash sequence, including the backslash character.</td>
</tr>
</tbody>
</table>

DESCRIPTION

The use of Tcl_Backslash is deprecated in favor of Tcl_UtfBackslash.

This is a utility procedure provided for backwards compatibility with non-internationalized Tcl extensions. It parses a backslash sequence and returns the low byte of the Unicode character corresponding to the sequence. Tcl_Backslash modifies *countPtr to contain the number of characters in the backslash sequence.
See the Tcl manual entry for information on the valid backslash sequences. All of the sequences described in the Tcl manual entry are supported by Tcl_Backslash.

SEE ALSO

Tcl, Tcl_UtfBackslash

KEYWORDS

backslash, parse

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tcl_GetInt, Tcl_GetDouble, Tcl_GetBoolean - convert from string to integer, double, or boolean

SYNOPSIS
#include <tcl.h>
int Tcl_GetInt(interp, src, intptr)
int Tcl_GetDouble(interp, src, doublePtr)
int Tcl_GetBoolean(interp, src, boolPtr)

ARGUMENTS
DESCRIPTION
KEYWORDS
**Tcl_Interp** *interp* (in)  
Interpreter to use for error reporting.

const char *src (in)  
Textual value to be converted.

int *intPtr (out)  
Points to place to store integer value converted from src.

double *doublePtr (out)  
Points to place to store double-precision floating-point value converted from src.

int *boolPtr (out)  
Points to place to store boolean value (0 or 1) converted from src.

---

**DESCRIPTION**

These procedures convert from strings to integers or double-precision floating-point values or booleans (represented as 0- or 1-valued integers). Each of the procedures takes a src argument, converts it to an internal form of a particular type, and stores the converted value at the location indicated by the procedure's third argument. If all goes well, each of the procedures returns TCL_OK. If src does not have the proper syntax for the desired type then TCL_ERROR is returned, an error message is left in the interpreter’s result, and nothing is stored at *intPtr or *doublePtr or *boolPtr.

**Tcl_GetInt** expects src to consist of a collection of integer digits, optionally signed and optionally preceded by white space. If the first two characters of src after the optional white space and sign are “0x” then src is expected to be in hexadecimal form; otherwise, if the first such character is “0” then src is expected to be in octal form; otherwise, src is
expected to be in decimal form.

Tcl_GetDouble expects src to consist of a floating-point number, which is: white space; a sign; a sequence of digits; a decimal point; a sequence of digits; the letter “e”; a signed decimal exponent; and more white space. Any of the fields may be omitted, except that the digits either before or after the decimal point must be present and if the “e” is present then it must be followed by the exponent number.

Tcl_GetBoolean expects src to specify a boolean value. If src is any of 0, false, no, or off, then Tcl_GetBoolean stores a zero value at *boolPtr. If src is any of 1, true, yes, or on, then 1 is stored at *boolPtr. Any of these values may be abbreviated, and upper-case spellings are also acceptable.

KEYWORDS

boolean, conversion, double, floating-point, integer
Tcl_RecordAndEvalObj - save command on history list before evaluating

**SYNOPSIS**

```c
#include <tcl.h>

int Tcl_RecordAndEvalObj(interp, cmdPtr, flags)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interp</td>
<td>Tcl interpreter in which to evaluate command.</td>
</tr>
<tr>
<td>cmdPtr</td>
<td>Points to a Tcl object containing a command (or sequence of commands) to execute.</td>
</tr>
<tr>
<td>flags</td>
<td>An OR'ed combination of flag bits. <strong>TCL_NO_EVAL</strong> means record the command but do not evaluate it. <strong>TCL_EVAL_GLOBAL</strong> means evaluate the command at global level instead of the current stack level.</td>
</tr>
</tbody>
</table>
Tcl_RecordAndEvalObj is invoked to record a command as an event on the history list and then execute it using Tcl_EvalObjEx (or Tcl_GlobalEvalObj if the TCL_EVAL_GLOBAL bit is set in flags). It returns a completion code such as TCL_OK just like Tcl_EvalObjEx, as well as a result object containing additional information (a result value or error message) that can be retrieved using Tcl_GetObjResult. If you do not want the command recorded on the history list then you should invoke Tcl_EvalObjEx instead of Tcl_RecordAndEvalObj. Normally Tcl_RecordAndEvalObj is only called with top-level commands typed by the user, since the purpose of history is to allow the user to re-issue recently invoked commands. If the flags argument contains the TCL_NO_EVAL bit then the command is recorded without being evaluated.

SEE ALSO

Tcl_EvalObjEx, Tcl_GetObjResult

KEYWORDS

command, event, execute, history, interpreter, object, record

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997 Sun Microsystems, Inc.
NAME
Tcl_CreateChannel, Tcl_GetChannelInstanceData,
Tcl_GetChannelType, Tcl_GetChannelName,
Tcl_GetChannelHandle, Tcl_GetChannelMode,
Tcl_GetChannelBufferSize, Tcl_SetChannelBufferSize,
Tcl_NotifyChannel, Tcl_BadChannelOption, Tcl_ChannelName,
Tcl_ChannelVersion, Tcl_ChannelBlockModeProc,
Tcl_ChannelCloseProc, Tcl_ChannelClose2Proc,
Tcl_ChannelInputProc, Tcl_ChannelOutputProc,
Tcl_ChannelSeekProc, Tcl_ChannelWideSeekProc,
Tcl_ChannelTruncateProc, Tcl_ChannelSetOptionProc,
Tcl_ChannelGetOptionProc, Tcl_ChannelWatchProc,
Tcl_ChannelGetHandleProc, Tcl_ChannelFlushProc,
Tcl_ChannelHandlerProc, Tcl_ChannelThreadActionProc,
Tcl_IsChannelShared, Tcl_IsChannelRegistered,
Tcl_CutChannel, Tcl_SpliceChannel, Tcl_IsChannelExisting,
Tcl_ClearChannelHandlers, Tcl_GetChannelThread,
Tcl_ChannelBuffered - procedures for creating and
manipulating channels

SYNOPSIS
#include <tcl.h>
Tcl_Channel
Tcl_CreateChannel(typePtr, channelName, instanceData,
mask)
ClientData
Tcl_GetChannelInstanceData(channel)
Tcl_ChannelType *
Tcl_GetChannelType(channel)
const char *
Tcl_GetChannelName(channel)
int
Tcl_GetChannelHandle(channel, direction, handlePtr)
Tcl_Threadld
Tcl_GetChannelThread(channel)
int
Tcl_GetChannelMode(channel)
int
Tcl_GetChannelBufferSize(channel)
Tcl_SetChannelBufferSize(channel, size)
Tcl_NotifyChannel(channel, mask)
int
Tcl_BadChannelOption(interp, optionName, optionList)
int
Tcl_IsChannelShared(channel)
int
Tcl_IsChannelRegistered(interp, channel)
int
Tcl_IsChannelExisting(channelName)
void
Tcl_CutChannel(channel)
void
Tcl_SpliceChannel(channel)
void
Tcl_ClearChannelHandlers(channel)
int
Tcl_ChannelBuffered(channel)
const char *
Tcl_ChannelName(typePtr)
Tcl_ChannelTypeVersion
Tcl_ChannelVersion(typePtr)
Tcl_DriverBlockModeProc *
Tcl_ChannelBlockModeProc(typePtr)
Tcl_DriverCloseProc *
Tcl_ChannelCloseProc(typePtr)
Tcl_DriverClose2Proc *
Tcl_ChannelClose2Proc(typePtr)
Tcl_DriverInputProc *
Tcl_ChannelInputProc(typePtr)
Tcl_DriverOutputProc *
Tcl_ChannelOutputProc(typePtr)
Tcl_DriverSeekProc *
Tcl_ChannelSeekProc(typePtr)
Tcl_DriverWideSeekProc *
Tcl_ChannelWideSeekProc(typePtr)
Tcl_DriverThreadActionProc *
Tcl_ChannelThreadActionProc(typePtr)
Tcl_DriverTruncateProc *
Tcl_ChannelTruncateProc(typePtr)
Tcl_DriverSetOptionProc *
Tcl_ChannelSetOptionProc(typePtr)
Tcl_DriverGetOptionProc *
Tcl_ChannelGetOptionProc(typePtr)
Tcl_DriverWatchProc *
Tcl_ChannelWatchProc(typePtr)
Tcl_DriverGetHandleProc *
Tcl_ChannelGetHandleProc(typePtr)
Tcl_DriverFlushProc *
Tcl_ChannelFlushProc(typePtr)
Tcl_DriverHandlerProc *
Tcl_ChannelHandlerProc(typePtr)

ARGUMENTS
DESCRIPTION
TCL_CHANNELTYPE
TYPENAME
VERSION
BLOCKMODEPROC
CLOSEPROC AND CLOSE2PROC
INPUTPROC
OUTPUTPROC
SEEKPROC AND WIDSEEKPROC
SETOPTIONPROC
GETOPTIONPROC
WATCHPROC
GETHANDLEPROC
FLUSHPROC
**HANDLERPROC**
**THREADACTIONPROC**
**TRUNCATEPROC**
**TCL_BADCHANNELOPTION**
**OLD CHANNEL TYPES**
**SEE ALSO**
**KEYWORDS**

**NAME**

Tcl_CreateChannel, Tcl_GetChannelInstanceData,
Tcl_GetChannelType, Tcl_GetChannelName, Tcl_GetChannelHandle,
Tcl_GetChannelMode, Tcl_GetChannelBufferSize,
Tcl_SetChannelBufferSize, Tcl_NotifyChannel, Tcl_BadChannelOption,
Tcl_ChannelName, Tcl_ChannelVersion, Tcl_ChannelBlockModeProc,
Tcl_ChannelCloseProc, Tcl_ChannelClose2Proc,
Tcl_ChannelInputProc, Tcl_ChannelOutputProc, Tcl_ChannelSeekProc,
Tcl_ChannelWideSeekProc, Tcl_ChannelTruncateProc,
Tcl_ChannelSetOptionProc, Tcl_ChannelGetOptionProc,
Tcl_ChannelWatchProc, Tcl_ChannelGetHandleProc,
Tcl_ChannelFlushProc, Tcl_ChannelHandlerProc,
Tcl_ChannelThreadActionProc, Tcl_IsChannelShared,
Tcl_IsChannelRegistered, Tcl_CutChannel, Tcl_SpliceChannel,
Tcl_IsChannelExisting, Tcl_ClearChannelHandlers,
Tcl_GetChannelThread, Tcl_ChannelBuffered - procedures for creating
and manipulating channels

**SYNOPSIS**

```
#include <tcl.h>

Tcl_Channel
Tcl_CreateChannel(typePtr, channelName, instanceData, mask)
ClientData
Tcl_GetChannelInstanceData(channel)
Tcl_ChannelType *
Tcl_GetChannelType(channel)
const char *
Tcl_GetChannelName(channel)
```
int
Tcl_GetChannelHandle(channel, direction, handlePtr)
Tcl_ThreadId
Tcl_GetChannelThreadId(channel)
int
Tcl_GetChannelMode(channel)
int
Tcl_GetChannelBufferSize(channel)
Tcl_SetChannelBufferSize(channel, size)
Tcl_NotifyChannel(channel, mask)
int
Tcl_BadChannelOption(interp, optionName, optionList)
int
Tcl_IsChannelShared(channel)
int
Tcl_IsChannelRegistered(interp, channel)
int
Tcl_IsChannelExisting(channelName)
void
Tcl_CutChannel(channel)
void
Tcl_SpliceChannel(channel)
void
Tcl_ClearChannelHandlers(channel)
int
Tcl_ChannelBuffered(channel)
const char *
Tcl_ChannelName(typePtr)
Tcl_ChannelTypeVersion
Tcl_ChannelVersion(typePtr)
Tcl_DriverBlockModeProc *
Tcl_ChannelBlockModeProc(typePtr)
Tcl_DriverCloseProc *
Tcl_ChannelCloseProc(typePtr)
Tcl_DriverClose2Proc *
Tcl_ChannelClose2Proc(typePtr)
Tcl_DriverInputProc *
const Tcl_ChannelType *typePtr (in)

Points to a structure containing the addresses of procedures that can be called to perform I/O and other functions on the channel.

const char *channelName (in)

The name of this channel, such as file3; must not be in use by any other channel. Can be NULL, in
which case the channel is created without a name.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientData <em>instanceData</em> (in)</td>
<td>Arbitrary one-word value to be associated with this channel. This value is passed to procedures in <code>typePtr</code> when they are invoked.</td>
</tr>
<tr>
<td>int <em>mask</em> (in)</td>
<td>OR-ed combination of <code>TCL_READABLE</code> and <code>TCL_WRITABLE</code> to indicate whether a channel is readable and writable.</td>
</tr>
<tr>
<td>Tcl_Channel <em>channel</em> (in)</td>
<td>The channel to operate on.</td>
</tr>
<tr>
<td>int <em>direction</em> (in)</td>
<td><code>TCL_READABLE</code> means the input handle is wanted; <code>TCL_WRITABLE</code> means the output handle is wanted.</td>
</tr>
<tr>
<td>ClientData <em>handlePtr</em> (out)</td>
<td>Points to the location where the desired OS-specific handle should be stored.</td>
</tr>
<tr>
<td>int <em>size</em> (in)</td>
<td>The size, in bytes, of buffers to allocate in this channel.</td>
</tr>
<tr>
<td>int <em>mask</em> (in)</td>
<td>An OR-ed combination of <code>TCL_READABLE</code>, <code>TCL_WRITABLE</code> and <code>TCL_EXCEPTION</code> that</td>
</tr>
</tbody>
</table>
indicates events that have occurred on this channel.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interpreter *interp (in)</code></td>
<td>Current interpreter. (can be NULL)</td>
</tr>
<tr>
<td>const char *optionName (in)</td>
<td>Name of the invalid option.</td>
</tr>
<tr>
<td>const char *optionList (in)</td>
<td>Specific options list (space separated words, without “-“) to append to the standard generic options list. Can be NULL for generic options error message only.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

Tcl uses a two-layered channel architecture. It provides a generic upper layer to enable C and Tcl programs to perform input and output using the same APIs for a variety of files, devices, sockets etc. The generic C APIs are described in the manual entry for `Tcl_OpenFileChannel`.

The lower layer provides type-specific channel drivers for each type of device supported on each platform. This manual entry describes the C APIs used to communicate between the generic layer and the type-specific channel drivers. It also explains how new types of channels can be added by providing new channel drivers.

Channel drivers consist of a number of components: First, each channel driver provides a `Tcl_ChannelType` structure containing pointers to functions implementing the various operations used by the generic layer to communicate with the channel driver. The `Tcl_ChannelType` structure and the functions referenced by it are described in the section `TCL_CHANNELTYPE`, below.

Second, channel drivers usually provide a Tcl command to create
instances of that type of channel. For example, the Tcl open command creates channels that use the file and command channel drivers, and the Tcl socket command creates channels that use TCP sockets for network communication.

Third, a channel driver optionally provides a C function to open channel instances of that type. For example, Tcl_OpenFileChannel opens a channel that uses the file channel driver, and Tcl_OpenTcpClient opens a channel that uses the TCP network protocol. These creation functions typically use Tcl_CreateChannel internally to open the channel.

To add a new type of channel you must implement a C API or a Tcl command that opens a channel by invoking Tcl_CreateChannel. When your driver calls Tcl_CreateChannel it passes in a Tcl_ChannelType structure describing the driver's I/O procedures. The generic layer will then invoke the functions referenced in that structure to perform operations on the channel.

Tcl_CreateChannel opens a new channel and associates the supplied typePtr and instanceData with it. The channel is opened in the mode indicated by mask. For a discussion of channel drivers, their operations and the Tcl_ChannelType structure, see the section TCL_CHANNELTYPE, below.

Tcl_CreateChannel interacts with the code managing the standard channels. Once a standard channel was initialized either through a call to Tcl_GetStdChannel or a call to Tcl_SetStdChannel closing this standard channel will cause the next call to Tcl_CreateChannel to make the new channel the new standard channel too. See Tcl_StandardChannels for a general treatise about standard channels and the behaviour of the Tcl library with regard to them.

Tcl_GetChannelInstanceData returns the instance data associated with the channel in channel. This is the same as the instanceData argument in the call to Tcl_CreateChannel that created this channel.

Tcl_GetChannelType returns a pointer to the Tcl_ChannelType
structure used by the channel in the *channel* argument. This is the same as the *typePtr* argument in the call to **Tcl_CreateChannel** that created this channel.

**Tcl_GetChannelName** returns a string containing the name associated with the channel, or NULL if the *channelName* argument to **Tcl_CreateChannel** was NULL.

**Tcl_GetChannelHandle** places the OS-specific device handle associated with *channel* for the given *direction* in the location specified by *handlePtr* and returns **TCL_OK**. If the channel does not have a device handle for the specified direction, then **TCL_ERROR** is returned instead. Different channel drivers will return different types of handle. Refer to the manual entries for each driver to determine what type of handle is returned.

**Tcl_GetChannelThread** returns the id of the thread currently managing the specified *channel*. This allows channel drivers to send their file events to the correct event queue even for a multi-threaded core.

**Tcl_GetChannelMode** returns an OR-ed combination of **TCL_READABLE** and **TCL_WRITABLE**, indicating whether the channel is open for input and output.

**Tcl_GetChannelBufferSize** returns the size, in bytes, of buffers allocated to store input or output in *channel*. If the value was not set by a previous call to **Tcl_SetChannelBufferSize**, described below, then the default value of 4096 is returned.

**Tcl_SetChannelBufferSize** sets the size, in bytes, of buffers that will be allocated in subsequent operations on the channel to store input or output. The *size* argument should be between ten and one million, allowing buffers of ten bytes to one million bytes. If *size* is outside this range, **Tcl_SetChannelBufferSize** sets the buffer size to 4096.

**Tcl_NotifyChannel** is called by a channel driver to indicate to the generic layer that the events specified by *mask* have occurred on the channel. Channel drivers are responsible for invoking this function
whenever the channel handlers need to be called for the channel. See \texttt{WATCHPROC} below for more details.

\textbf{Tcl\_BadChannelOption} is called from driver specific \texttt{setOptionProc} or \texttt{getOptionProc} to generate a complete error message.

\textbf{Tcl\_ChannelBuffered} returns the number of bytes of input currently buffered in the internal buffer (push back area) of the channel itself. It does not report about the data in the overall buffers for the stack of channels the supplied channel is part of.

\textbf{Tcl\_IsChannelShared} checks the refcount of the specified \textit{channel} and returns whether the \textit{channel} was shared among multiple interpreters (result == 1) or not (result == 0).

\textbf{Tcl\_IsChannelRegistered} checks whether the specified \textit{channel} is registered in the given \textit{interpreter} (result == 1) or not (result == 0).

\textbf{Tcl\_IsChannelExisting} checks whether a channel with the specified name is registered in the (thread)-global list of all channels (result == 1) or not (result == 0).

\textbf{Tcl\_CutChannel} removes the specified \textit{channel} from the (thread)global list of all channels (of the current thread). Application to a channel still registered in some interpreter is not allowed. Also notifies the driver if the \textbf{Tcl\_ChannelType} version is \texttt{TCL\_CHANNEL\_VERSION\_4} (or higher), and \textbf{Tcl\_DriverThreadActionProc} is defined for it.

\textbf{Tcl\_SpliceChannel} adds the specified \textit{channel} to the (thread)global list of all channels (of the current thread). Application to a channel registered in some interpreter is not allowed. Also notifies the driver if the \textbf{Tcl\_ChannelType} version is \texttt{TCL\_CHANNEL\_VERSION\_4} (or higher), and \textbf{Tcl\_DriverThreadActionProc} is defined for it.

\textbf{Tcl\_ClearChannelHandlers} removes all channelhandlers and event scripts associated with the specified \textit{channel}, thus shutting down all event processing for this channel.
A channel driver provides a `Tcl_ChannelType` structure that contains pointers to functions that implement the various operations on a channel; these operations are invoked as needed by the generic layer. The structure was versioned starting in Tcl 8.3.2/8.4 to correct a problem with stacked channel drivers. See the OLD CHANNEL TYPES section below for details about the old structure.

The `Tcl_ChannelType` structure contains the following fields:

```c
typedef struct Tcl_ChannelType {
    char *typeName;
    Tcl_ChannelTypeVersion version;
    Tcl_DriverCloseProc *closeProc;
    Tcl_DriverInputProc *inputProc;
    Tcl_DriverOutputProc *outputProc;
    Tcl_DriverSeekProc *seekProc;
    Tcl_DriverSetOptionProc *setOptionProc;
    Tcl_DriverGetOptionProc *getOptionProc;
    Tcl_DriverWatchProc *watchProc;
    Tcl_DriverGetHandleProc *getHandleProc;
    Tcl_DriverClose2Proc *close2Proc;
    Tcl_DriverBlockModeProc *blockModeProc;
    Tcl_DriverFlushProc *flushProc;
    Tcl_DriverHandlerProc *handlerProc;
    Tcl_DriverWideSeekProc *wideSeekProc;
    Tcl_DriverThreadActionProc *threadActionProc;
    Tcl_DriverTruncateProc *truncateProc;
} Tcl_ChannelType;
```

It is not necessary to provide implementations for all channel operations. Those which are not necessary may be set to NULL in the struct: `blockModeProc`, `seekProc`, `setOptionProc`, `getOptionProc`, and `close2Proc`, in addition to `flushProc`, `handlerProc`, `threadActionProc`, and `wideSeekProc`.
and \textit{truncateProc}. Other functions that cannot be implemented in a meaningful way should return \texttt{EINVAL} when called, to indicate that the operations they represent are not available. Also note that \textit{wideSeekProc} can be \texttt{NULL} if \textit{seekProc} is.

The user should only use the above structure for \texttt{Tcl\_ChannelType} instantiation. When referencing fields in a \texttt{Tcl\_ChannelType} structure, the following functions should be used to obtain the values: \texttt{Tcl\_ChannelName}, \texttt{Tcl\_ChannelVersion}, \texttt{Tcl\_ChannelBlockModeProc}, \texttt{Tcl\_ChannelCloseProc}, \texttt{Tcl\_ChannelClose2Proc}, \texttt{Tcl\_ChannelInputProc}, \texttt{Tcl\_ChannelOutputProc}, \texttt{Tcl\_ChannelSeekProc}, \texttt{Tcl\_ChannelWideSeekProc}, \texttt{Tcl\_ChannelThreadActionProc}, \texttt{Tcl\_ChannelTruncateProc}, \texttt{Tcl\_ChannelSetOptionProc}, \texttt{Tcl\_ChannelGetOptionProc}, \texttt{Tcl\_ChannelWatchProc}, \texttt{Tcl\_ChannelGetHandleProc}, \texttt{Tcl\_ChannelFlushProc}, or \texttt{Tcl\_ChannelHandlerProc}.

The change to the structures was made in such a way that standard channel types are binary compatible. However, channel types that use stacked channels (i.e. TLS, Trf) have new versions to correspond to the above change since the previous code for stacked channels had problems.

\textbf{TYPENAME}

The \textit{typeName} field contains a null-terminated string that identifies the type of the device implemented by this driver, e.g. file or socket.

This value can be retrieved with \texttt{Tcl\_ChannelName}, which returns a pointer to the string.

\textbf{VERSION}

The \textit{version} field should be set to the version of the structure that you require. \texttt{TCL\_CHANNEL\_VERSION\_2} is the minimum recommended. \texttt{TCL\_CHANNEL\_VERSION\_3} must be set to specify the
wideSeekProc member. **TCL_CHANNEL_VERSION_4** must be set to specify the threadActionProc member (includes wideSeekProc). **TCL_CHANNEL_VERSION_5** must be set to specify the truncateProc members (includes wideSeekProc and threadActionProc). If it is not set to any of these, then this Tcl_ChannelType is assumed to have the original structure. See **OLD CHANNEL TYPES** for more details. While Tcl will recognize and function with either structures, stacked channels must be of at least **TCL_CHANNEL_VERSION_2** to function correctly.

This value can be retrieved with **Tcl_ChannelVersion**, which returns one of **TCL_CHANNEL_VERSION_5**, **TCL_CHANNEL_VERSION_4**, **TCL_CHANNEL_VERSION_3**, **TCL_CHANNEL_VERSION_2** or **TCL_CHANNEL_VERSION_1**.

**BLOCKMODEPROC**

The blockModeProc field contains the address of a function called by the generic layer to set blocking and nonblocking mode on the device. **BlockModeProc** should match the following prototype:

```c
typedef int Tcl_DriverBlockModeProc(
    ClientData instanceData,
    int mode);
```

The instanceData is the same as the value passed to **Tcl_CreateChannel** when this channel was created. The mode argument is either **TCL_MODE_BLOCKING** or **TCL_MODE_NONBLOCKING** to set the device into blocking or nonblocking mode. The function should return zero if the operation was successful, or a nonzero POSIX error code if the operation failed.

If the operation is successful, the function can modify the supplied instanceData to record that the channel entered blocking or nonblocking mode and to implement the blocking or nonblocking behavior. For some device types, the blocking and nonblocking behavior can be implemented by the underlying operating system; for other device
This value can be retrieved with **Tcl\_ChannelBlockModeProc**, which returns a pointer to the function.

A channel driver **not** supplying a **blockModeProc** has to be very, very careful. It has to tell the generic layer exactly which blocking mode is acceptable to it, and should this also document for the user so that the blocking mode of the channel is not changed to an inacceptable value. Any confusion here may lead the interpreter into a (spurious and difficult to find) deadlock.

---

**CLOSEPROC AND CLOSE2PROC**

The **closeProc** field contains the address of a function called by the generic layer to clean up driver-related information when the channel is closed. **CloseProc** must match the following prototype:

```c
typedef int Tcl_DriverCloseProc(
    ClientData instanceData,
    Tcl_Interp *interp);
```

The **instanceData** argument is the same as the value provided to **Tcl\_CreateChannel** when the channel was created. The function should release any storage maintained by the channel driver for this channel, and close the input and output devices encapsulated by this channel. All queued output will have been flushed to the device before this function is called, and no further driver operations will be invoked on this instance after calling the **closeProc**. If the close operation is successful, the procedure should return zero; otherwise it should return a nonzero POSIX error code. In addition, if an error occurs and **interp** is not NULL, the procedure should store an error message in the interpreter's result.

Alternatively, channels that support closing the read and write sides independently may set **closeProc** to **TCL\_CLOSE2PROC** and set
close2Proc to the address of a function that matches the following prototype:

```c
typedef int Tcl_DriverClose2Proc(
    ClientData instanceData,
    Tcl_Interp  *interp,
    int flags);
```

The close2Proc will be called with flags set to an OR’ed combination of TCL_CLOSE_READ or TCL_CLOSE_WRITE to indicate that the driver should close the read and/or write side of the channel. The channel driver may be invoked to perform additional operations on the channel after close2Proc is called to close one or both sides of the channel. If flags is 0 (zero), the driver should close the channel in the manner described above for closeProc. No further operations will be invoked on this instance after close2Proc is called with all flags cleared. In all cases, the close2Proc function should return zero if the close operation was successful; otherwise it should return a nonzero POSIX error code. In addition, if an error occurs and interp is not NULL, the procedure should store an error message in the interpreter’s result.

The closeProc and close2Proc values can be retrieved with Tcl_ChannelCloseProc or Tcl_ChannelClose2Proc, which return a pointer to the respective function.

**INPUTPROC**

The inputProc field contains the address of a function called by the generic layer to read data from the file or device and store it in an internal buffer. InputProc must match the following prototype:

```c
typedef int Tcl_DriverInputProc(
    ClientData instanceData,
    char  *buf,
    int  bufSize,
    ...);
```
InstanceData is the same as the value passed to Tcl_CreateChannel when the channel was created. The buf argument points to an array of bytes in which to store input from the device, and the bufSize argument indicates how many bytes are available at buf.

The errorCodePtr argument points to an integer variable provided by the generic layer. If an error occurs, the function should set the variable to a POSIX error code that identifies the error that occurred.

The function should read data from the input device encapsulated by the channel and store it at buf. On success, the function should return a nonnegative integer indicating how many bytes were read from the input device and stored at buf. On error, the function should return -1. If an error occurs after some data has been read from the device, that data is lost.

If inputProc can determine that the input device has some data available but less than requested by the bufSize argument, the function should only attempt to read as much data as is available and return without blocking. If the input device has no data available whatsoever and the channel is in nonblocking mode, the function should return an EAGAIN error. If the input device has no data available whatsoever and the channel is in blocking mode, the function should block for the shortest possible time until at least one byte of data can be read from the device; then, it should return as much data as it can read without blocking.

This value can be retrieved with Tcl_ChannelInputProc, which returns a pointer to the function.

**OUTPUTPROC**

The outputProc field contains the address of a function called by the generic layer to transfer data from an internal buffer to the output device. OutputProc must match the following prototype:
typedef int Tcl_DriverOutputProc(
    ClientData instanceData,
    const char *buf,
    int toWrite,
    int *errorCodePtr);

InstanceData is the same as the value passed to Tcl_CreateChannel when the channel was created. The buf argument contains an array of bytes to be written to the device, and the toWrite argument indicates how many bytes are to be written from the buf argument.

The errorCodePtr argument points to an integer variable provided by the generic layer. If an error occurs, the function should set this variable to a POSIX error code that identifies the error.

The function should write the data at buf to the output device encapsulated by the channel. On success, the function should return a nonnegative integer indicating how many bytes were written to the output device. The return value is normally the same as toWrite, but may be less in some cases such as if the output operation is interrupted by a signal. If an error occurs the function should return -1. In case of error, some data may have been written to the device.

If the channel is nonblocking and the output device is unable to absorb any data whatsoever, the function should return -1 with an EAGAIN error without writing any data.

This value can be retrieved with Tcl_ChannelOutputProc, which returns a pointer to the function.

SEEKPROC AND WIDSEEKPROC

The seekProc field contains the address of a function called by the generic layer to move the access point at which subsequent input or output operations will be applied. SeekProc must match the following prototype:
typedef int Tcl_DriverSeekProc(
    ClientData instanceData,
    long offset,
    int seekMode,
    int *errorCodePtr);

The *instanceData* argument is the same as the value given to *Tcl_CreateChannel* when this channel was created. *Offset* and *seekMode* have the same meaning as for the *Tcl_Seek* procedure (described in the manual entry for *Tcl_OpenFileChannel*).

The *errorCodePtr* argument points to an integer variable provided by the generic layer for returning *errno* values from the function. The function should set this variable to a POSIX error code if an error occurs. The function should store an *EINVAL* error code if the channel type does not implement seeking.

The return value is the new access point or -1 in case of error. If an error occurred, the function should not move the access point.

If there is a non-NULL *seekProc* field, the *wideSeekProc* field may contain the address of an alternative function to use which handles wide (i.e. larger than 32-bit) offsets, so allowing seeks within files larger than 2GB. The *wideSeekProc* will be called in preference to the *seekProc*, but both must be defined if the *wideSeekProc* is defined. *WideSeekProc* must match the following prototype:

```
typedef Tcl_WideInt Tcl_DriverWideSeekProc(
    ClientData instanceData,
    Tcl_WideInt offset,
    int seekMode,
    int *errorCodePtr);
```

The arguments and return values mean the same thing as with *seekProc* above, except that the type of offsets and the return type are
different.

The `seekProc` value can be retrieved with `Tcl_ChannelSeekProc`, which returns a pointer to the function, and similarly the `wideSeekProc` can be retrieved with `Tcl_ChannelWideSeekProc`.

**SETOPTIONPROC**

The `setOptionProc` field contains the address of a function called by the generic layer to set a channel type specific option on a channel. `setOptionProc` must match the following prototype:

```c
typedef int Tcl_DriverSetOptionProc(
    ClientData instanceData,
    Tcl_Interp *interp,
    const char *optionName,
    const char *newValue);
```

`optionName` is the name of an option to set, and `newValue` is the new value for that option, as a string. The `instanceData` is the same as the value given to `Tcl_CreateChannel` when this channel was created. The function should do whatever channel type specific action is required to implement the new value of the option.

Some options are handled by the generic code and this function is never called to set them, e.g. `-blockmode`. Other options are specific to each channel type and the `setOptionProc` procedure of the channel driver will get called to implement them. The `setOptionProc` field can be NULL, which indicates that this channel type supports no type specific options.

If the option value is successfully modified to the new value, the function returns `TCL_OK`. It should call `Tcl_BadChannelOption` which itself returns `TCL_ERROR` if the `optionName` is unrecognized. If `newValue` specifies a value for the option that is not supported or if a system call error occurs, the function should leave an error message in
the result field of interp if interp is not NULL. The function should also call Tcl_SetErrno to store an appropriate POSIX error code.

This value can be retrieved with Tcl_ChannelSetOptionProc, which returns a pointer to the function.

### GETOPTIONPROC

The getOptionProc field contains the address of a function called by the generic layer to get the value of a channel type specific option on a channel. getOptionProc must match the following prototype:

```c
typedef int Tcl_DriverGetOptionProc(
    ClientData instanceData,
    Tcl_Interp *interp,
    const char *optionName,
    Tcl_DString *optionValue);
```

OptionName is the name of an option supported by this type of channel. If the option name is not NULL, the function stores its current value, as a string, in the Tcl dynamic string optionValue. If optionName is NULL, the function stores in optionValue an alternating list of all supported options and their current values. On success, the function returns TCL_OK. It should call Tcl_BadChannelOption which itself returns TCL_ERROR if the optionName is unrecognized. If a system call error occurs, the function should leave an error message in the result of interp if interp is not NULL. The function should also call Tcl_SetErrno to store an appropriate POSIX error code.

Some options are handled by the generic code and this function is never called to retrieve their value, e.g. -blockmode. Other options are specific to each channel type and the getOptionProc procedure of the channel driver will get called to implement them. The getOptionProc field can be NULL, which indicates that this channel type supports no type specific options.
This value can be retrieved with `Tcl_ChannelGetOptionProc`, which returns a pointer to the function.

**WATCHPROC**

The `watchProc` field contains the address of a function called by the generic layer to initialize the event notification mechanism to notice events of interest on this channel. `WatchProc` should match the following prototype:

```c
typedef void Tcl_DriverWatchProc(
    ClientData instanceData,
    int mask);
```

The `instanceData` is the same as the value passed to `Tcl_CreateChannel` when this channel was created. The `mask` argument is an OR-ed combination of `TCL_READABLE`, `TCL_WRITABLE` and `TCL_EXCEPTION`; it indicates events the caller is interested in noticing on this channel.

The function should initialize device type specific mechanisms to notice when an event of interest is present on the channel. When one or more of the designated events occurs on the channel, the channel driver is responsible for calling `Tcl_NotifyChannel` to inform the generic channel module. The driver should take care not to starve other channel drivers or sources of callbacks by invoking `Tcl_NotifyChannel` too frequently. Fairness can be insured by using the Tcl event queue to allow the channel event to be scheduled in sequence with other events. See the description of `Tcl_QueueEvent` for details on how to queue an event.

This value can be retrieved with `Tcl_ChannelWatchProc`, which returns a pointer to the function.

**GETHANDLEPROC**

The `getHandleProc` field contains the address of a function called by the
generic layer to retrieve a device-specific handle from the channel. 
*GetHandleProc* should match the following prototype:

```c
typedef int Tcl_DriverGetHandleProc(
    ClientData instanceData,
    int direction,
    ClientData *handlePtr);
```

*InstanceData* is the same as the value passed to *Tcl_CreateChannel* when this channel was created. The *direction* argument is either *TCL_READABLE* to retrieve the handle used for input, or *TCL_WRITABLE* to retrieve the handle used for output.

If the channel implementation has device-specific handles, the function should retrieve the appropriate handle associated with the channel, according the *direction* argument. The handle should be stored in the location referred to by *handlePtr*, and *TCL_OK* should be returned. If the channel is not open for the specified direction, or if the channel implementation does not use device handles, the function should return *TCL_ERROR*.

This value can be retrieved with *Tcl_ChannelGetHandleProc*, which returns a pointer to the function.

---

**FLUSHPROC**

The *flushProc* field is currently reserved for future use. It should be set to NULL. *FlushProc* should match the following prototype:

```c
typedef int Tcl_DriverFlushProc(
    ClientData instanceData);
```

This value can be retrieved with *Tcl_ChannelFlushProc*, which returns a pointer to the function.
The *handlerProc* field contains the address of a function called by the generic layer to notify the channel that an event occurred. It should be defined for stacked channel drivers that wish to be notified of events that occur on the underlying (stacked) channel. *HandlerProc* should match the following prototype:

```
typedef int Tcl_DriverHandlerProc(
    ClientData instanceData,
    int interestMask);
```

*InstanceData* is the same as the value passed to *Tcl_CreateChannel* when this channel was created. The *interestMask* is an OR-ed combination of *TCL_READABLE* or *TCL_WRITABLE*; it indicates what type of event occurred on this channel.

This value can be retrieved with *Tcl_ChannelHandlerProc*, which returns a pointer to the function.

The *threadActionProc* field contains the address of the function called by the generic layer when a channel is created, closed, or going to move to a different thread, i.e. whenever thread-specific driver state might have to be initialized or updated. It can be NULL. The action *TCL_CHANNEL_THREAD_REMOVE* is used to notify the driver that it should update or remove any thread-specific data it might be maintaining for the channel.

The action *TCL_CHANNEL_THREAD_INSERT* is used to notify the driver that it should update or initialize any thread-specific data it might be maintaining using the calling thread as the associate. See *Tcl_CutChannel* and *Tcl_SpliceChannel* for more detail.
typedef void Tcl_DriverThreadActionProc(
    ClientData instanceData,
    int action);

InstanceData is the same as the value passed to Tcl_CreateChannel when this channel was created.

These values can be retrieved with Tcl_ChannelThreadActionProc, which returns a pointer to the function.

TRUNCATEPROC

The truncateProc field contains the address of the function called by the generic layer when a channel is truncated to some length. It can be NULL.

typedef int Tcl_DriverTruncateProc(
    ClientData instanceData,
    Tcl_WideInt length);

InstanceData is the same as the value passed to Tcl_CreateChannel when this channel was created, and length is the new length of the underlying file, which should not be negative. The result should be 0 on success or an errno code (suitable for use with Tcl_SetErrno) on failure.

These values can be retrieved with Tcl_ChannelTruncateProc, which returns a pointer to the function.

TCL_BADCHANNELOPTION

This procedure generates a “bad option” error message in an (optional) interpreter. It is used by channel drivers when an invalid Set/Get option is requested. Its purpose is to concatenate the generic options list to the specific ones and factorize the generic options error message string.
It always returns **TCL_ERROR**

An error message is generated in *interp*'s result object to indicate that a command was invoked with a bad option. The message has the form

```
bad option "blah": should be one of
  <...generic options...>+<...specific options...>
```

so you get for instance:

```
bad option "-blah": should be one of -blocking,
  -buffering, -buffersize, -eofchar, -translation,
  -peername, or -sockname
```

when called with *optionList* equal to “peername sockname”

“blah” is the *optionName* argument and “<specific options>” is a space separated list of specific option words. The function takes good care of inserting minus signs before each option, commas after, and an “or” before the last option.

### OLD CHANNEL TYPES

The original (8.3.1 and below) **Tcl_ChannelType** structure contains the following fields:

```c
typedef struct Tcl_ChannelType {
    char *typeName;
    Tcl_DriverBlockModeProc *blockModeProc;
    Tcl_DriverCloseProc *closeProc;
    Tcl_DriverInputProc *inputProc;
    Tcl_DriverOutputProc *outputProc;
    Tcl_DriverSeekProc *seekProc;
} Tcl_ChannelType;
```
It is still possible to create channel with the above structure. The internal channel code will determine the version. It is imperative to use the new `Tcl_ChannelType` structure if you are creating a stacked channel driver, due to problems with the earlier stacked channel implementation (in 8.2.0 to 8.3.1).

Prior to 8.4.0 (i.e. during the later releases of 8.3 and early part of the 8.4 development cycle) the `Tcl_ChannelType` structure contained the following fields:

```c
typedef struct Tcl_ChannelType {
    char *typeName;
    Tcl_ChannelTypeVersion version;
    Tcl_DriverCloseProc *closeProc;
    Tcl_DriverInputProc *inputProc;
    Tcl_DriverOutputProc *outputProc;
    Tcl_DriverSeekProc *seekProc;
    Tcl_DriverSetOptionProc *setOptionProc;
    Tcl_DriverGetOptionProc *getOptionProc;
    Tcl_DriverWatchProc *watchProc;
    Tcl_DriverGetHandleProc *getHandleProc;
    Tcl_DriverClose2Proc *close2Proc;
    Tcl_DriverBlockModeProc *blockModeProc;
    Tcl_DriverFlushProc *flushProc;
    Tcl_DriverHandlerProc *handlerProc;
    Tcl_DriverTruncateProc *truncateProc;
} Tcl_ChannelType;
```
When the above structure is registered as a channel type, the version field should always be `TCL_CHANNEL_VERSION_2`.

**SEE ALSO**

`tcl_close`, `tcl_openfilechannel`, `tcl_seterrno`, `tcl_queueevent`, `tcl_stackchannel`, `tcl_getstdchannel`

**KEYWORDS**

blocking, channel driver, channel registration, channel type, nonblocking

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996-1997 Sun Microsystems, Inc.
Copyright © 1997-2000 Ajuba Solutions.
NAME

Tcl_NewBooleanObj, Tcl_SetBooleanObj,
Tcl_GetBooleanFromObj - store/retrieve boolean value in a
Tcl_Obj

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_NewBooleanObj(boolValue)
Tcl_SetBooleanObj(objPtr, boolValue)
int
Tcl_GetBooleanFromObj(interp, objPtr, boolPtr)

ARGUMENTS

DESCRIPTION

SEE ALSO

KEYWORDS

NAME

Tcl_NewBooleanObj, Tcl_SetBooleanObj, Tcl_GetBooleanFromObj - store/retrieve boolean value in a Tcl_Obj

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_NewBooleanObj(boolValue)
Tcl_SetBooleanObj(objPtr, boolValue)
int
Tcl_GetBooleanFromObj(interp, objPtr, boolPtr)

ARGUMENTS
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <code>boolValue</code> (in)</td>
<td>Integer value to be stored as a boolean value in a Tcl_Obj.</td>
</tr>
<tr>
<td>Tcl_Obj *<code>objPtr</code> (in/out)</td>
<td>Points to the Tcl_Obj in which to store, or from which to retrieve a boolean value.</td>
</tr>
<tr>
<td>Tcl_Interp *<code>interp</code> (in/out)</td>
<td>If a boolean value cannot be retrieved, an error message is left in the interpreter's result object unless <code>interp</code> is NULL.</td>
</tr>
<tr>
<td>int *<code>boolPtr</code> (out)</td>
<td>Points to place where <code>Tcl_GetBooleanFromObj</code> stores the boolean value (0 or 1) obtained from <code>objPtr</code>.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

These procedures are used to pass boolean values to and from Tcl as Tcl_Obj's. When storing a boolean value into a Tcl_Obj, any non-zero integer value in `boolValue` is taken to be the boolean value 1, and the integer value 0 is taken to be the boolean value 0.

`Tcl_NewBooleanObj` creates a new Tcl_Obj, stores the boolean value `boolValue` in it, and returns a pointer to the new Tcl_Obj. The new Tcl_Obj has reference count of zero.

`Tcl_SetBooleanObj` accepts `objPtr`, a pointer to an existing Tcl_Obj, and stores in the Tcl_Obj *`objPtr` the boolean value `boolValue`. This is a write operation on *`objPtr`, so `objPtr` must be unshared. Attempts to write to a shared Tcl_Obj will panic. A successful write of `boolValue` into
*objPtr implies the freeing of any former value stored in *objPtr.

Tcl_GetBooleanFromObj attempts to retrieve a boolean value from the value stored in *objPtr. If objPtr holds a string value recognized by Tcl_GetBoolean, then the recognized boolean value is written at the address given by boolPtr. If objPtr holds any value recognized as a number by Tcl, then if that value is zero a 0 is written at the address given by boolPtr and if that value is non-zero a 1 is written at the address given by boolPtr. In all cases where a value is written at the address given by boolPtr, Tcl_GetBooleanFromObj returns TCL_OK. If the value of objPtr does not meet any of the conditions above, then TCL_ERROR is returned and an error message is left in the interpreter's result unless interp is NULL. Tcl_GetBooleanFromObj may also make changes to the internal fields of *objPtr so that future calls to Tcl_GetBooleanFromObj on the same objPtr can be performed more efficiently.

Note that the routines Tcl_GetBooleanFromObj and Tcl_GetBoolean are not functional equivalents. The set of values for which Tcl_GetBooleanFromObj will return TCL_OK is strictly larger than the set of values for which Tcl_GetBoolean will do the same. For example, the value “5” passed to Tcl_GetBooleanFromObj will lead to a TCL_OK return (and the boolean value 1), while the same value passed to Tcl_GetBoolean will lead to a TCL_ERROR return.

SEE ALSO

Tcl_NewObj, Tcl_IsShared, Tcl_GetBoolean

KEYWORDS

boolean, object
NAME
Tcl_RegExpMatch, Tcl_RegExpCompile, Tcl_RegExpExec,
Tcl_RegExpRange, Tcl_GetRegExpFromObj,
Tcl_RegExpMatchObj, Tcl_RegExpExecObj,
Tcl_RegExpGetInfo - Pattern matching with regular
expressions

SYNOPSIS
#include <tcl.h>
int
Tcl_RegExpMatchObj(interp, textObj, patObj)
int
Tcl_RegExpMatch(interp, text, pattern)
Tcl_RegExp
Tcl_RegExpCompile(interp, pattern)
int
Tcl_RegExpExec(interp, regexp, text, start)
void
Tcl_RegExpRange(regexp, index, startPtr, endPtr)
Tcl_RegExp
Tcl_GetRegExpFromObj(interp, patObj, cflags)
int
Tcl_RegExpExecObj(interp, regexp, textObj, offset,
nmatches, eflags)
void
Tcl_RegExpGetInfo(regexp, infoPtr)

ARGUMENTS
DESCRIPTION
TCL_REG_ADVANCED
TCL_REG_EXTENDED
TCL_REG_BASIC
TCL_REG_EXPANDED
**TCL_REG_QUOTE**
**TCL_REG_NOCASE**
**TCL_REG_NEWLINE**
**TCL_REG_NLSTOP**
**TCL_REG_NLANCH**
**TCL_REG_NOSUB**
**TCL_REG_CANMATCH**
**TCL_REG_NOTBOL**
**TCL_REG_NOTEOL**

**SEE ALSO**
**KEYWORDS**

**NAME**
Tcl_RegExpMatch, Tcl_RegExpCompile, Tcl_RegExpExec,
Tcl_RegExpRange, Tcl_GetRegExpFromObj, Tcl_RegExpMatchObj,
Tcl_RegExpExecObj, Tcl_RegExpGetInfo - Pattern matching with
regular expressions

**SYNOPSIS**

```
#include <tcl.h>

int Tcl_RegExpMatchObj(interp, textObj, patObj)
int Tcl_RegExpMatch(interp, text, pattern)
Tcl_RegExp Tcl_RegExpCompile(interp, pattern)
int Tcl_RegExpExec(interp, regexp, text, start)
void Tcl_RegExpRange(regexp, index, startPtr, endPtr)
Tcl_RegExp Tcl_GetRegExpFromObj(interp, patObj, cflags)
int Tcl_RegExpExecObj(interp, regexp, textObj, offset, nmatches, eflags)
void Tcl_RegExpGetInfo(regexp, infoPtr)
```
**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interp *interp</code></td>
<td><code>Tcl</code> interpreter to use for error reporting. The interpreter may be NULL if no error reporting is desired.</td>
</tr>
<tr>
<td><code>Tcl_Obj *textObj</code></td>
<td>Refers to the object from which to get the text to search. The internal representation of the object may be converted to a form that can be efficiently searched.</td>
</tr>
<tr>
<td><code>Tcl_Obj *patObj</code></td>
<td>Refers to the object from which to get a regular expression. The compiled regular expression is cached in the object.</td>
</tr>
<tr>
<td><code>char *text</code></td>
<td>Text to search for a match with a regular expression.</td>
</tr>
<tr>
<td><code>const char *pattern</code></td>
<td>String in the form of a regular expression pattern.</td>
</tr>
<tr>
<td><code>Tcl_RegExp *regexp</code></td>
<td>Compiled regular expression. Must have been returned previously by <code>Tcl_GetRegExpFromObj</code> or <code>Tcl_RegExpCompile</code>.</td>
</tr>
<tr>
<td><code>char *start</code></td>
<td>If <code>text</code> is just a portion of the text to search, the starting position within the text must be specified.</td>
</tr>
</tbody>
</table>
some other string, this argument identifies the beginning of the larger string. If it is not the same as *text*, then no “^” matches will be allowed.

<table>
<thead>
<tr>
<th>Description</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <code>index</code> (in)</td>
<td>Specifies which range is desired: 0 means the range of the entire match, 1 or greater means the range that matched a parenthesized sub-expression.</td>
</tr>
<tr>
<td>const char **<code>startPtr</code> (out)</td>
<td>The address of the first character in the range is stored here, or NULL if there is no such range.</td>
</tr>
<tr>
<td>const char **<code>endPtr</code> (out)</td>
<td>The address of the character just after the last one in the range is stored here, or NULL if there is no such range.</td>
</tr>
<tr>
<td>int <code>cflags</code> (in)</td>
<td>OR-ed combination of the compilation flags TCL_REG_ADVANCED, TCL_REG_EXTENDED, TCL_REG_BASIC, TCL_REG_EXPANDED, TCL_REG_QUOTE, TCL_REG_NOCASE, TCL_REG_NEWLINE, TCL_REG_NLSTOP, TCL_REG_NLANCH,</td>
</tr>
</tbody>
</table>
### int offset (in)

The character offset into the text where matching should begin. The value of the offset has no impact on ^ matches. This behavior is controlled by eflags.

### int nmatches (in)

The number of matching subexpressions that should be remembered for later use. If this value is 0, then no subexpression match information will be computed. If the value is -1, then all of the matching subexpressions will be remembered. Any other value will be taken as the maximum number of subexpressions to remember.

### int eflags (in)

OR-ed combination of the execution flags TCL_REG_NOTBOL and TCL_REG_NOTEOL. See below for more information.

### Tcl_RegExpInfo *infoPtr (out)

The address of the location where information about a previous match

TCL_REG_NOSUB, and TCL_REG_CANMATCH. See below for more information.
DESCRIPTION

**Tcl_RegExpMatch** determines whether its *pattern* argument matches *regexp*, where *regexp* is interpreted as a regular expression using the rules in the **re_syntax** reference page. If there is a match then **Tcl_RegExpMatch** returns 1. If there is no match then **Tcl_RegExpMatch** returns 0. If an error occurs in the matching process (e.g. *pattern* is not a valid regular expression) then **Tcl_RegExpMatch** returns -1 and leaves an error message in the interpreter result.

**Tcl_RegExpMatchObj** is similar to **Tcl_RegExpMatch** except it operates on the Tcl objects *textObj* and *patObj* instead of UTF strings. **Tcl_RegExpMatchObj** is generally more efficient than **Tcl_RegExpMatch**, so it is the preferred interface.

**Tcl_RegExpCompile**, **Tcl_RegExpExec**, and **Tcl_RegExpRange** provide lower-level access to the regular expression pattern matcher. **Tcl_RegExpCompile** compiles a regular expression string into the internal form used for efficient pattern matching. The return value is a token for this compiled form, which can be used in subsequent calls to **Tcl_RegExpExec** or **Tcl_RegExpRange**. If an error occurs while compiling the regular expression then **Tcl_RegExpCompile** returns NULL and leaves an error message in the interpreter result. Note: the return value from **Tcl_RegExpCompile** is only valid up to the next call to **Tcl_RegExpCompile**; it is not safe to retain these values for long periods of time.

**Tcl_RegExpExec** executes the regular expression pattern matcher. It returns 1 if *text* contains a range of characters that match *regexp*, 0 if no match is found, and -1 if an error occurs. In the case of an error, **Tcl_RegExpExec** leaves an error message in the interpreter result. When searching a string for multiple matches of a pattern, it is important to distinguish between the start of the original string and the start of the current search. For example, when searching for the second occurrence of a match, the *text* argument might point to the character
just after the first match; however, it is important for the pattern matcher to know that this is not the start of the entire string, so that it does not allow "^" atoms in the pattern to match. The start argument provides this information by pointing to the start of the overall string containing text. Start will be less than or equal to text; if it is less than text then no ^ matches will be allowed.

**Tcl_RegExpRange** may be invoked after **Tcl_RegExpExec** returns; it provides detailed information about what ranges of the string matched what parts of the pattern. **Tcl_RegExpRange** returns a pair of pointers in *startPtr and *endPtr that identify a range of characters in the source string for the most recent call to **Tcl_RegExpExec**. Index indicates which of several ranges is desired: if index is 0, information is returned about the overall range of characters that matched the entire pattern; otherwise, information is returned about the range of characters that matched the index'th parenthesized subexpression within the pattern. If there is no range corresponding to index then NULL is stored in *startPtr and *endPtr.

**Tcl_GetRegExpFromObj**, **Tcl_RegExpExecObj**, and **Tcl_RegExpGetInfo** are object interfaces that provide the most direct control of Henry Spencer's regular expression library. For users that need to modify compilation and execution options directly, it is recommended that you use these interfaces instead of calling the internal regexp functions. These interfaces handle the details of UTF to Unicode translations as well as providing improved performance through caching in the pattern and string objects.

**Tcl_GetRegExpFromObj** attempts to return a compiled regular expression from the patObj. If the object does not already contain a compiled regular expression it will attempt to create one from the string in the object and assign it to the internal representation of the patObj. The return value of this function is of type **Tcl_RegExp**. The return value is a token for this compiled form, which can be used in subsequent calls to **Tcl_RegExpExecObj** or **Tcl_RegExpGetInfo**. If an error occurs while compiling the regular expression then **Tcl_GetRegExpFromObj** returns NULL and leaves an error message in the interpreter result. The regular expression token can be used as
long as the internal representation of patObj refers to the compiled form. The cflags argument is a bit-wise OR of zero or more of the following flags that control the compilation of patObj:

**TCL_REG_ADVANCED**
Compile advanced regular expressions (“ARE”s). This mode corresponds to the normal regular expression syntax accepted by the Tcl `regexp` and `regsub` commands.

**TCL_REG_EXTENDED**
Compile extended regular expressions (“ERE”s). This mode corresponds to the regular expression syntax recognized by Tcl 8.0 and earlier versions.

**TCL_REG_BASIC**
Compile basic regular expressions (“BRE”s). This mode corresponds to the regular expression syntax recognized by common Unix utilities like `sed` and `grep`. This is the default if no flags are specified.

**TCL_REG_EXPANDED**
Compile the regular expression (basic, extended, or advanced) using an expanded syntax that allows comments and whitespace. This mode causes non-backslashed non-bracket-expression white space and #-to-end-of-line comments to be ignored.

**TCL_REG_QUOTE**
Compile a literal string, with all characters treated as ordinary characters.

**TCL_REG_NOCASE**
Compile for matching that ignores upper/lower case distinctions.

**TCL_REG_NEWLINE**
Compile for newline-sensitive matching. By default, newline is a completely ordinary character with no special meaning in either regular expressions or strings. With this flag, “[^” bracket expressions and “.” never match newline, “^” matches an empty
string after any newline in addition to its normal function, and “$” matches an empty string before any newline in addition to its normal function. REG_NEWLINE is the bit-wise OR of REG_NLSTOP and REG_NLANCH.

TCL_REG_NLSTOP
Compile for partial newline-sensitive matching, with the behavior of “[^]” bracket expressions and “.” affected, but not the behavior of “^” and “$”. In this mode, “[^]” bracket expressions and “.” never match newline.

TCL_REG_NLANCH
Compile for inverse partial newline-sensitive matching, with the behavior of “^” and “$” (the “anchors”) affected, but not the behavior of “[^]” bracket expressions and “.”. In this mode “^” matches an empty string after any newline in addition to its normal function, and “$” matches an empty string before any newline in addition to its normal function.

TCL_REG_NOSUB
Compile for matching that reports only success or failure, not what was matched. This reduces compile overhead and may improve performance. Subsequent calls to Tcl_RegExpGetInfo or Tcl_RegExpRange will not report any match information.

TCL_REG_CANMATCH
Compile for matching that reports the potential to complete a partial match given more text (see below).

Only one of TCL_REG_EXTENDED, TCL_REG_ADVANCED, TCL_REG_BASIC, and TCL_REG_QUOTE may be specified.

Tcl_RegExpExecObj executes the regular expression pattern matcher. It returns 1 if objPtr contains a range of characters that match regexp, 0 if no match is found, and -1 if an error occurs. In the case of an error, Tcl_RegExpExecObj leaves an error message in the interpreter result. The nmatches value indicates to the matcher how many subexpressions are of interest. If nmatches is 0, then no subexpression
match information is recorded, which may allow the matcher to make various optimizations. If the value is -1, then all of the subexpressions in the pattern are remembered. If the value is a positive integer, then only that number of subexpressions will be remembered. Matching begins at the specified Unicode character index given by offset. Unlike Tcl_RegExpExec, the behavior of anchors is not affected by the offset value. Instead the behavior of the anchors is explicitly controlled by the eflags argument, which is a bit-wise OR of zero or more of the following flags:

**TCL_REG_NOTBOL**
The starting character will not be treated as the beginning of a line or the beginning of the string, so “^” will not match there. Note that this flag has no effect on how “\A” matches.

**TCL_REG_NOTEOL**
The last character in the string will not be treated as the end of a line or the end of the string, so “$” will not match there. Note that this flag has no effect on how “\Z” matches.

Tcl_RegExpGetInfo retrieves information about the last match performed with a given regular expression regexp. The infoPtr argument contains a pointer to a structure that is defined as follows:

```c
typedef struct Tcl_RegExpInfo {
    int nsubs;
    Tcl_RegExpIndices *matches;
    long extendStart;
} Tcl_RegExpInfo;
```

The nsubs field contains a count of the number of parenthesized subexpressions within the regular expression. If the TCL_REG_NOSUB was used, then this value will be zero. The matches field points to an array of nsubs values that indicate the bounds of each subexpression matched. The first element in the array refers to the range matched by the entire regular expression, and subsequent elements refer to the
parenthesized subexpressions in the order that they appear in the pattern. Each element is a structure that is defined as follows:

```c
typedef struct Tcl_RegExpIndices {
    long start;
    long end;
} Tcl_RegExpIndices;
```

The `start` and `end` values are Unicode character indices relative to the offset location within `objPtr` where matching began. The `start` index identifies the first character of the matched subexpression. The `end` index identifies the first character after the matched subexpression. If the subexpression matched the empty string, then `start` and `end` will be equal. If the subexpression did not participate in the match, then `start` and `end` will be set to -1.

The `extendStart` field in `Tcl_RegExpInfo` is only set if the `TCL_REG_CANMATCH` flag was used. It indicates the first character in the string where a match could occur. If a match was found, this will be the same as the beginning of the current match. If no match was found, then it indicates the earliest point at which a match might occur if additional text is appended to the string. If it is no match is possible even with further text, this field will be set to -1.

**SEE ALSO**

`re_syntax`

**KEYWORDS**

match, pattern, regular expression, string, subexpression, `Tcl_RegExpIndices`, `Tcl_RegExpInfo`
NAME

Tcl_CallWhenDeleted, Tcl_DontCallWhenDeleted - Arrange for callback when interpreter is deleted

SYNOPSIS

#include <tcl.h>
Tcl_CallWhenDeleted(interp, proc, clientData)
Tcl_DontCallWhenDeleted(interp, proc, clientData)

ARGUMENTS

Tcl_Interp *interp (in)  Interpreter with which to associated callback.
Tcl_InterpDeleteProc *proc (in)  Procedure to call when interp is deleted.
ClientData clientData (in)  Arbitrary one-word value to pass to proc.

DESCRIPTION

Tcl_CallWhenDeleted arranges for proc to be called by Tcl_DeleteInterp if/when interp is deleted at some future time. Proc will be invoked just before the interpreter is deleted, but the interpreter will still be valid at the time of the call. Proc should have arguments and result that match the type Tcl_InterpDeleteProc:
typedef void Tcl_InterpDeleteProc(
    ClientData clientData, 
    Tcl_Interp *interp);

The *clientData* and *interp* parameters are copies of the *clientData* and *interp* arguments given to *Tcl_CallWhenDeleted*. Typically, *clientData* points to an application-specific data structure that *proc* uses to perform cleanup when an interpreter is about to go away. *Proc* does not return a value.

*Tcl_DontCallWhenDeleted* cancels a previous call to *Tcl_CallWhenDeleted* with the same arguments, so that *proc* will not be called after all when *interp* is deleted. If there is no deletion callback that matches *interp*, *proc*, and *clientData* then the call to *Tcl_DontCallWhenDeleted* has no effect.

**KEYWORDS**

callback, delete, interpreter

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tcl_NewByteArrayObj, Tcl_SetByteArrayObj, Tcl_GetByteArrayFromObj, Tcl_SetByteArrayLength - manipulate Tcl objects as a arrays of bytes

SYNOPSIS
#include <tcl.h>
Tcl_Obj *
Tcl_NewByteArrayObj(bytes, length)
void
Tcl_SetByteArrayObj(objPtr, bytes, length)
unsigned char *
Tcl_GetByteArrayFromObj(objPtr, lengthPtr)
unsigned char *
Tcl_SetByteArrayLength(objPtr, length)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS
**unsigned char**

**Tcl_GetByteArrayFromObj**(objPtr, lengthPtr)

**unsigned char**

**Tcl_SetByteArrayLength**(objPtr, length)

### ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const unsigned char *bytes (in)</td>
<td>The array of bytes used to initialize or set a byte-array object.</td>
</tr>
<tr>
<td>int length (in)</td>
<td>The length of the array of bytes. It must be &gt;= 0.</td>
</tr>
<tr>
<td>Tcl_Obj *objPtr (in/out)</td>
<td>For <strong>Tcl_SetByteArrayObj</strong>, this points to the object to be converted to byte-array type. For <strong>Tcl_GetByteArrayFromObj</strong> and <strong>Tcl_SetByteArrayLength</strong>, this points to the object from which to get the byte-array value; if objPtr does not already point to a byte-array object, it will be converted to one.</td>
</tr>
<tr>
<td>int *lengthPtr (out)</td>
<td>If non-NULL, filled with the length of the array of bytes in the object.</td>
</tr>
</tbody>
</table>

### DESCRIPTION

These procedures are used to create, modify, and read Tcl byte-array objects from C code. Byte-array objects are typically used to hold the
results of binary IO operations or data structures created with the `binary` command. In Tcl, an array of bytes is not equivalent to a string. Conceptually, a string is an array of Unicode characters, while a byte-array is an array of 8-bit quantities with no implicit meaning. Accessor functions are provided to get the string representation of a byte-array or to convert an arbitrary object to a byte-array. Obtaining the string representation of a byte-array object (by calling `Tcl_GetStringFromObj`) produces a properly formed UTF-8 sequence with a one-to-one mapping between the bytes in the internal representation and the UTF-8 characters in the string representation.

`Tcl_NewByteArrayObj` and `Tcl_SetByteArrayObj` will create a new object of byte-array type or modify an existing object to have a byte-array type. Both of these procedures set the object's type to be byte-array and set the object's internal representation to a copy of the array of bytes given by `bytes`. `Tcl_NewByteArrayObj` returns a pointer to a newly allocated object with a reference count of zero. `Tcl_SetByteArrayObj` invalidates any old string representation and, if the object is not already a byte-array object, frees any old internal representation.

`Tcl_GetByteArrayFromObj` converts a Tcl object to byte-array type and returns a pointer to the object's new internal representation as an array of bytes. The length of this array is stored in `lengthPtr` if `lengthPtr` is non-NULL. The storage for the array of bytes is owned by the object and should not be freed. The contents of the array may be modified by the caller only if the object is not shared and the caller invalidates the string representation.

`Tcl_SetByteArrayLength` converts the Tcl object to byte-array type and changes the length of the object's internal representation as an array of bytes. If `length` is greater than the space currently allocated for the array, the array is reallocated to the new length; the newly allocated bytes at the end of the array have arbitrary values. If `length` is less than the space currently allocated for the array, the length of array is reduced to the new length. The return value is a pointer to the object's new array of bytes.
**NAME**

Tcl_DoWhenIdle, Tcl_CancelIdleCall - invoke a procedure when there are no pending events

**SYNOPSIS**

```c
#include <tcl.h>
Tcl_DoWhenIdle(proc, clientData)
Tcl_CancelIdleCall(proc, clientData)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_IdleProc *proc (in)</td>
<td>Procedure to invoke.</td>
</tr>
<tr>
<td>ClientData clientData (in)</td>
<td>Arbitrary one-word value to pass to proc.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

*Tcl_DoWhenIdle* arranges for *proc* to be invoked when the application becomes idle. The application is considered to be idle when *Tcl_DoOneEvent* has been called, could not find any events to handle, and is about to go to sleep waiting for an event to occur. At this point all pending *Tcl_DoWhenIdle* handlers are invoked. For each call to *Tcl_DoWhenIdle* there will be a single call to *proc*; after *proc* is invoked the handler is automatically removed. *Tcl_DoWhenIdle* is only usable in programs that use *Tcl_DoOneEvent* to dispatch events.

*Proc* should have arguments and result that match the type *Tcl_IdleProc*:
typedef void Tcl_IdleProc(ClientData clientData);

The clientData parameter to proc is a copy of the clientData argument given to Tcl_DoWhenIdle. Typically, clientData points to a data structure containing application-specific information about what proc should do.

Tcl_CancelIdleCall may be used to cancel one or more previous calls to Tcl_DoWhenIdle: if there is a Tcl_DoWhenIdle handler registered for proc and clientData, then it is removed without invoking it. If there is more than one handler on the idle list that refers to proc and clientData, all of the handlers are removed. If no existing handlers match proc and clientData then nothing happens.

Tcl_DoWhenIdle is most useful in situations where (a) a piece of work will have to be done but (b) it is possible that something will happen in the near future that will change what has to be done or require something different to be done. Tcl_DoWhenIdle allows the actual work to be deferred until all pending events have been processed. At this point the exact work to be done will presumably be known and it can be done exactly once.

For example, Tcl_DoWhenIdle might be used by an editor to defer display updates until all pending commands have been processed. Without this feature, redundant redispaly might occur in some situations, such as the processing of a command file.

BUGS

At present it is not safe for an idle callback to reschedule itself continuously. This will interact badly with certain features of Tk that attempt to wait for all idle callbacks to complete. If you would like for an idle callback to reschedule itself continuously, it is better to use a timer handler with a zero timeout period.

KEYWORDS
callback, defer, idle callback
NAME
Tcl_SetChannelError, Tcl_SetChannelErrorInterp,
Tcl_GetChannelError, Tcl_GetChannelErrorInterp - functions to
create/intercept Tcl errors by channel drivers.

SYNOPSIS
#include <tcl.h>
void
Tcl_SetChannelError(chan, msg)
void
Tcl_SetChannelErrorInterp(interp, msg)
void
Tcl_GetChannelError(chan, msgPtr)
void
Tcl_GetChannelErrorInterp(interp, msgPtr)

ARGUMENTS

DESCRIPTION
Tcl_DriverCloseProc
Tcl_DriverInputProc
Tcl_DriverOutputProc
Tcl_DriverSeekProc
Tcl_DriverWideSeekProc
Tcl_DriverSetOptionProc
Tcl_DriverGetOptionProc
Tcl_DriverWatchProc
Tcl_DriverBlockModeProc
Tcl_DriverGetHandleProc
Tcl_DriverHandlerProc
Tcl_StackChannel
Tcl_Seek
Tcl_Tell
Tcl_ReadRaw
Tcl_Read
Tcl_ReadChars
Tcl_Gets
Tcl_GetsObj
Tcl_Flush
Tcl_WriteRaw
Tcl_WriteObj
Tcl_Write
Tcl_WriteChars
Tcl_Close
Tcl_UnregisterChannel
Tcl_UnstackChannel

SEE ALSO
KEYWORDS

NAME

Tcl_SetChannelError, Tcl_SetChannelErrorInterp, Tcl_GetChannelError, Tcl_GetChannelErrorInterp - functions to create/intercept Tcl errors by channel drivers.

SYNOPSIS

#include <tcl.h>
void
Tcl_SetChannelError(chan, msg)
void
Tcl_SetChannelErrorInterp(interp, msg)
void
Tcl_GetChannelError(chan, msgPtr)
void
Tcl_GetChannelErrorInterp(interp, msgPtr)

ARGUMENTS

Tcl_Channel chan (in) Refers to the Tcl channel whose bypass area is accessed.
**Tcl_Interp** *(in)*

Refers to the Tcl interpreter whose bypass area is accessed.

**Tcl_Obj** *(in)*

Error message put into a bypass area. A list of return options and values, followed by a string message. Both message and the option/value information are optional.

**Tcl_Obj** *(out)*

Reference to a place where the message stored in the accessed bypass area can be stored in.

### DESCRIPTION

The current definition of a Tcl channel driver does not permit the direct return of arbitrary error messages, except for the setting and retrieval of channel options. All other functions are restricted to POSIX error codes.

The functions described here overcome this limitation. Channel drivers are allowed to use **Tcl_SetChannelError** and **Tcl_SetChannelErrorInterp** to place arbitrary error messages in bypass areas defined for channels and interpreters. And the generic I/O layer uses **Tcl_GetChannelError** and **Tcl_GetChannelErrorInterp** to look for messages in the bypass areas and arrange for their return as errors. The posix error codes set by a driver are used now if and only if no messages are present.

**Tcl_SetChannelError** stores error information in the bypass area of the specified channel. The number of references to the **msg** object goes up by one. Previously stored information will be discarded, by releasing the reference held by the channel. The channel reference must not be
Tcl_SetChannelErrorInterp stores error information in the bypass area of the specified interpreter. The number of references to the msg object goes up by one. Previously stored information will be discarded, by releasing the reference held by the interpreter. The interpreter reference must not be NULL.

Tcl_GetChannelError places either the error message held in the bypass area of the specified channel into msgPtr, or NULL; and resets the bypass. I.e. after an invokation all following invokations will return NULL, until an intervening invokation of Tcl_SetChannelError with a non-NULL message. The msgPtr must not be NULL. The reference count of the message is not touched. The reference previously held by the channel is now held by the caller of the function and it is its responsibility to release that reference when it is done with the object.

Tcl_GetChannelErrorInterp places either the error message held in the bypass area of the specified interpreter into msgPtr, or NULL; and resets the bypass. I.e. after an invokation all following invokations will return NULL, until an intervening invokation of Tcl_SetChannelErrorInterp with a non-NULL message. The msgPtr must not be NULL. The reference count of the message is not touched. The reference previously held by the interpreter is now held by the caller of the function and it is its responsibility to release that reference when it is done with the object.

Which functions of a channel driver are allowed to use which bypass function is listed below, as is which functions of the public channel API may leave a messages in the bypass areas.

Tcl_DriverCloseProc
- May use Tcl_SetChannelErrorInterp, and only this function.

Tcl_DriverInputProc
- May use Tcl_SetChannelError, and only this function.

Tcl_DriverOutputProc
May use **Tcl_SetChannelError**, and only this function.

**Tcl_DriverSeekProc**
May use **Tcl_SetChannelError**, and only this function.

**Tcl_DriverWideSeekProc**
May use **Tcl_SetChannelError**, and only this function.

**Tcl_DriverSetOptionProc**
Has already the ability to pass arbitrary error messages. Must **not** use any of the new functions.

**Tcl_DriverGetOptionProc**
Has already the ability to pass arbitrary error messages. Must **not** use any of the new functions.

**Tcl_DriverWatchProc**
Must **not** use any of the new functions. Is internally called and has no ability to return any type of error whatsoever.

**Tcl_DriverBlockModeProc**
May use **Tcl_SetChannelError**, and only this function.

**Tcl_DriverGetHandleProc**
Must **not** use any of the new functions. It is only a low-level function, and not used by Tcl commands.

**Tcl_DriverHandlerProc**
Must **not** use any of the new functions. Is internally called and has no ability to return any type of error whatsoever.

Given the information above the following public functions of the Tcl C API are affected by these changes. I.e. when these functions are called the channel may now contain a stored arbitrary error message requiring processing by the caller.

**Tcl_STACKChannel**

**Tcl_Seed**
Tcl_Tell
Tcl_ReadRaw
Tcl_Read
Tcl_ReadChars
Tcl_Gets
Tcl_GetsObj
Tcl_FLUSH
Tcl_WriteRaw
Tcl_WriteObj
Tcl_Write
Tcl_WriteChars

All other API functions are unchanged. Especially the functions below leave all their error information in the interpreter result.

Tcl_Close
Tcl_UnregisterChannel
Tcl_UnstackChannel

SEE ALSO

Tcl_Close, Tcl_OpenFileChannel, Tcl_SetErrno

KEYWORDS

channel driver, error messages, channel type

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
Tcl_RegisterConfig - procedures to register embedded configuration information

SYNOPSIS
#include <tcl.h>
void
Tcl_RegisterConfig(interp, pkgName, configuration, valEncoding)

ARGUMENTS

DESCRIPTION

(1)
(2)
::pkgName::pkgconfig list
::pkgName::pkgconfig get key

TCL_CONFIG

KEYWORDS
**DESCRIPTION**

The function described here has its base in TIP 59 and provides extensions with support for the embedding of configuration information into their binary library and the generation of a Tcl-level interface for...

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const char <strong>pkgName</strong> (in)</td>
<td>Contains the name of the package registering the embedded configuration as ASCII string. This means that this information is in UTF-8 too. Must not be NULL.</td>
</tr>
<tr>
<td>Tcl_Config <strong>configuration</strong> (in)</td>
<td>Refers to an array of Tcl_Config entries containing the information embedded in the binary library. Must not be NULL. The end of the array is signaled by either a key identical to NULL, or a key referring to the empty string.</td>
</tr>
<tr>
<td>const char <strong>valEncoding</strong> (in)</td>
<td>Contains the name of the encoding used to store the configuration values as ASCII string. This means that this information is in UTF-8 too. Must not be NULL.</td>
</tr>
</tbody>
</table>
To embed configuration information into their binary library an extension has to define a non-volatile array of Tcl_Config entries in one of its source files and then call **Tcl_RegisterConfig** to register that information.

**Tcl_RegisterConfig** takes four arguments; first, a reference to the interpreter we are registering the information with, second, the name of the package registering its configuration information, third, a pointer to an array of structures, and fourth a string declaring the encoding used by the configuration values.

The string *valEncoding* contains the name of an encoding known to Tcl. All these names are use only characters in the ASCII subset of UTF-8 and are thus implicitly in the UTF-8 encoding. It is expected that keys are legible English text and therefore using the ASCII subset of UTF-8. In other words, they are expected to be in UTF-8 too. The values associated with the keys can be any string however. For these the contents of *valEncoding* define which encoding was used to represent the characters of the strings.

Each element of the *configuration* array refers to two strings containing the key and the value associated with that key. The end of the array is signaled by either an empty key or a key identical to NULL. The function makes **no** copy of the *configuration* array. This means that the caller has to make sure that the memory holding this array is never released. This is the meaning behind the word **non-volatile** used earlier. The easiest way to accomplish this is to define a global static array of Tcl_Config entries. See the file “generic/tclPkgConfig.c” in the sources of the Tcl core for an example.

When called **Tcl_RegisterConfig** will

\[(1)\]

- create a namespace having the provided *pkgName*, if not yet existing.
create the command `pkgconfig` in that namespace and link it to the provided information so that the keys from \_configuration\_ and their associated values can be retrieved through calls to `pkgconfig`.

The command `pkgconfig` will provide two subcommands, `list` and `get`:

```
::pkgName::pkgconfig list
   Returns a list containing the names of all defined keys.

::pkgName::pkgconfig get key
   Returns the configuration value associated with the specified `key`.
```

**TCL_CONFIG**

The **Tcl_Config** structure contains the following fields:

```c
typedef struct Tcl_Config {
    const char* key;
    const char* value;
} Tcl_Config;
```

**KEYWORDS**

embedding, configuration, binary library

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2002 Andreas Kupries <andreas_kupries(at)users.sourceforge.net>
NAME
Tcl_SaveInterpState, Tcl_RestoreInterpState,
Tcl_DiscardInterpState, Tcl_SaveResult, Tcl_RestoreResult,
Tcl_DiscardResult - save and restore an interpreter's state

SYNOPSIS
#include <tcl.h>
Tcl_InterpState
Tcl_SaveInterpState(interp, status)
int
Tcl_RestoreInterpState(interp, state)
Tcl_DiscardInterpState(state)
Tcl_SaveResult(interp, savedPtr)
Tcl_RestoreResult(interp, savedPtr)
Tcl_DiscardResult(savedPtr)

ARGUMENTS
DESCRIPTION
KEYWORDS
Tcl_SaveResult(*interp, savedPtr)
Tcl_RestoreResult(*interp, savedPtr)
Tcl_DiscardResult(savedPtr)

ARGUMENTS

Tcl_Interp *interp (in)  Interpreter for which state should be saved.
int status (in)  Return code value to save as part of interpreter state.
Tcl_InterpState state (in)  Saved state token to be restored or discarded.
Tcl_SavedResult *savedPtr (in)  Pointer to location where interpreter result should be saved or restored.

DESCRIPTION

These routines allows a C procedure to take a snapshot of the current state of an interpreter so that it can be restored after a call to Tcl_Eval or some other routine that modifies the interpreter state. There are two triplets of routines meant to work together.

The first triplet stores the snapshot of interpreter state in an opaque token returned by Tcl_SaveInterpState. That token value may then be passed back to one of Tcl_RestoreInterpState or Tcl_DiscardInterpState, depending on whether the interp state is to be restored. So long as one of the latter two routines is called, Tcl will take care of memory management.

The second triplet stores the snapshot of only the interpreter result (not its complete state) in memory allocated by the caller. These routines are passed a pointer to a Tcl_SavedResult structure that is used to store enough information to restore the interpreter result. This structure can
be allocated on the stack of the calling procedure. These routines do not save the state of any error information in the interpreter (e.g. the -errorcode or -errorinfo return options, when an error is in progress).

Because the routines `Tcl_SaveInterpState`, `Tcl_RestoreInterpState`, and `Tcl_DiscardInterpState` perform a superset of the functions provided by the other routines, any new code should only make use of the more powerful routines. The older, weaker routines `Tcl_SaveResult`, `Tcl_RestoreResult`, and `Tcl_DiscardResult` continue to exist only for the sake of existing programs that may already be using them.

`Tcl_SaveInterpState` takes a snapshot of those portions of interpreter state that make up the full result of script evaluation. This include the interpreter result, the return code (passed in as the status argument, and any return options, including -errorinfo and -errorcode when an error is in progress. This snapshot is returned as an opaque token of type `Tcl_InterpState`. The call to `Tcl_SaveInterpState` does not itself change the state of the interpreter. Unlike `Tcl_SaveResult`, it does not reset the interpreter.

`Tcl_RestoreInterpState` accepts a `Tcl_InterpState` token previously returned by `Tcl_SaveInterpState` and restores the state of the interp to the state held in that snapshot. The return value of `Tcl_RestoreInterpState` is the status value originally passed to `Tcl_SaveInterpState` when the snapshot token was created.

`Tcl_DiscardInterpState` is called to release a `Tcl_InterpState` token previously returned by `Tcl_SaveInterpState` when that snapshot is not to be restored to an interp.

The `Tcl_InterpState` token returned by `Tcl_SaveInterpState` must eventually be passed to either `Tcl_RestoreInterpState` or `Tcl_DiscardInterpState` to avoid a memory leak. Once the `Tcl_InterpState` token is passed to one of them, the token is no longer valid and should not be used anymore.

`Tcl_SaveResult` moves the string and object results of `interp` into the
location specified by statePtr. Tcl_SaveResult clears the result for interp and leaves the result in its normal empty initialized state.

Tcl_RestoreResult moves the string and object results from statePtr back into interp. Any result or error that was already in the interpreter will be cleared. The statePtr is left in an uninitialized state and cannot be used until another call to Tcl_SaveResult.

Tcl_DiscardResult releases the saved interpreter state stored at statePtr. The state structure is left in an uninitialized state and cannot be used until another call to Tcl_SaveResult.

Once Tcl_SaveResult is called to save the interpreter result, either Tcl_RestoreResult or Tcl_DiscardResult must be called to properly clean up the memory associated with the saved state.
NAME
Tcl_CreateObjCommand, Tcl_DeleteCommand,
Tcl_DeleteCommandFromToken, Tcl_GetCommandInfo,
Tcl_GetCommandInfoFromToken, Tcl_SetCommandInfo,
Tcl_SetCommandInfoFromToken, Tcl_GetCommandName,
Tcl_GetCommandFullName, Tcl_GetCommandFromObj -
implement new commands in C

SYNOPSIS
#include <tcl.h>
Tcl_Command
Tcl_CreateObjCommand(interp, cmdName, proc, clientData,
deleteProc)
int
Tcl_DeleteCommand(interp, cmdName)
int
Tcl_DeleteCommandFromToken(interp, token)
int
Tcl_GetCommandInfo(interp, cmdName, infoPtr)
int
Tcl_SetCommandInfo(interp, cmdName, infoPtr)
int
Tcl_GetCommandInfoFromToken(token, infoPtr)
int
Tcl_SetCommandInfoFromToken(token, infoPtr)
const char *
Tcl_GetCommandName(interp, token)
void
Tcl_GetCommandFullName(interp, token, objPtr)
Tcl_Command
Tcl_GetCommandFromObj(interp, objPtr)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS

NAME

Tcl_CreateObjCommand, Tcl_DeleteCommand,
Tcl_DeleteCommandFromToken, Tcl_GetCommandInfo,
Tcl_GetCommandInfoFromToken, Tcl_SetCommandInfo,
Tcl_SetCommandInfoFromToken, Tcl_GetCommandName,
Tcl_GetCommandFullName, Tcl_GetCommandFromObj - implement
new commands in C

SYNOPSIS

#include <tcl.h>
Tcl_Command
Tcl_CreateObjCommand(interp, cmdName, proc, clientData, deleteProc)
int
Tcl_DeleteCommand(interp, cmdName)
int
Tcl_DeleteCommandFromToken(interp, token)
int
Tcl_GetCommandInfo(interp, cmdName, infoPtr)
int
Tcl_SetCommandInfo(interp, cmdName, infoPtr)
int
Tcl_GetCommandInfoFromToken(token, infoPtr)
int
Tcl_SetCommandInfoFromToken(token, infoPtr)
const char *
Tcl_GetCommandName(interp, token)
void
Tcl_GetCommandFullName(interp, token, objPtr)
Tcl_Command
Tcl_GetCommandFromObj(interp, objPtr)
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_Interp <em>interp</em></em> (in)</td>
<td>Interpreter in which to create a new command or that contains a command.</td>
</tr>
<tr>
<td>char *cmdName (in)</td>
<td>Name of command.</td>
</tr>
<tr>
<td>*<em>Tcl_ObjCmdProc <em>proc</em></em> (in)</td>
<td>Implementation of the new command: <em>proc</em> will be called whenever <em>cmdName</em> is invoked as a command.</td>
</tr>
<tr>
<td>ClientData <strong>clientData</strong> (in)</td>
<td>Arbitrary one-word value to pass to <em>proc</em> and <em>deleteProc</em>.</td>
</tr>
<tr>
<td>*<em>Tcl_CmdDeleteProc <em>deleteProc</em></em> (in)</td>
<td>Procedure to call before <em>cmdName</em> is deleted from the interpreter; allows for command-specific cleanup. If NULL, then no procedure is called before the command is deleted.</td>
</tr>
<tr>
<td><strong>Tcl_Command token</strong> (in)</td>
<td>Token for command, returned by previous call to <strong>Tcl_CreateObjCommand</strong>. The command must not have been deleted.</td>
</tr>
<tr>
<td>*<em>Tcl_CmdInfo <em>infoPtr</em></em> (in/out)</td>
<td>Pointer to structure containing various information about a Tcl command.</td>
</tr>
</tbody>
</table>
Tcl_Obj *objPtr (in) Object containing the name of a Tcl command.

**DESCRIPTION**

**Tcl_CreateObjCommand** defines a new command in *interp* and associates it with procedure *proc* such that whenever *name* is invoked as a Tcl command (e.g., via a call to *Tcl_EvalObjEx*) the Tcl interpreter will call *proc* to process the command.

**Tcl_CreateObjCommand** deletes any existing command *name* already associated with the interpreter (however see below for an exception where the existing command is not deleted). It returns a token that may be used to refer to the command in subsequent calls to **Tcl_GetCommandName**. If *name* contains any :: namespace qualifiers, then the command is added to the specified namespace; otherwise the command is added to the global namespace. If **Tcl_CreateObjCommand** is called for an interpreter that is in the process of being deleted, then it does not create a new command and it returns NULL. *proc* should have arguments and result that match the type **Tcl_ObjCmdProc**:

```c
typedef int Tcl_ObjCmdProc(
    ClientData clientData,
    Tcl_Interp *interp,
    int objc,
    Tcl_Obj *const objv[]);
```

When *proc* is invoked, the *clientData* and *interp* parameters will be copies of the *clientData* and *interp* arguments given to **Tcl_CreateObjCommand**. Typically, *clientData* points to an application-specific data structure that describes what to do when the command procedure is invoked. *objc* and *objv* describe the arguments to the command, *objc* giving the number of argument objects (including the command name) and *objv* giving the values of the arguments. The *objv*
array will contain objc values, pointing to the argument objects. Unlike argv[argv] used in a string-based command procedure, objv[objc] will not contain NULL.

Additionally, when proc is invoked, it must not modify the contents of the objv array by assigning new pointer values to any element of the array (for example, objv[2] = NULL) because this will cause memory to be lost and the runtime stack to be corrupted. The const in the declaration of objv will cause ANSI-compliant compilers to report any such attempted assignment as an error. However, it is acceptable to modify the internal representation of any individual object argument. For instance, the user may call Tcl_GetIntFromObj on objv[2] to obtain the integer representation of that object; that call may change the type of the object that objv[2] points at, but will not change where objv[2] points.

proc must return an integer code that is either TCL_OK, TCL_ERROR, TCL_RETURN, TCL_BREAK, or TCL_CONTINUE. See the Tcl overview man page for details on what these codes mean. Most normal commands will only return TCL_OK or TCL_ERROR. In addition, if proc needs to return a non-empty result, it can call Tcl_SetObjResult to set the interpreter's result. In the case of a TCL_OK return code this gives the result of the command, and in the case of TCL_ERROR this gives an error message. Before invoking a command procedure, Tcl_EvalObjEx sets interpreter's result to point to an object representing an empty string, so simple commands can return an empty result by doing nothing at all.

The contents of the objv array belong to Tcl and are not guaranteed to persist once proc returns: proc should not modify them. Call Tcl_SetObjResult if you want to return something from the objv array.

Ordinarily, Tcl_CreateObjCommand deletes any existing command name already associated with the interpreter. However, if the existing command was created by a previous call to Tcl_CreateCommand, Tcl_CreateObjCommand does not delete the command but instead arranges for the Tcl interpreter to call the Tcl_ObjCmdProc proc in the future. The old string-based Tcl_CmdProc associated with the
command is retained and its address can be obtained by subsequent
Tcl_GetCommandInfo calls. This is done for backwards compatibility.

*DeleteProc* will be invoked when (if) *name* is deleted. This can occur
through a call to Tcl_DeleteCommand,
Tcl_DeleteCommandFromToken, or Tcl_DeleteInterp, or by replacing
*name* in another call to Tcl_CreateObjCommand. *DeleteProc* is
invoked before the command is deleted, and gives the application an
opportunity to release any structures associated with the command.
*DeleteProc* should have arguments and result that match the type
Tcl_CmdDeleteProc:

```c
typedef void Tcl_CmdDeleteProc(
    ClientData clientData);
```

The *clientData* argument will be the same as the *clientData* argument
passed to Tcl_CreateObjCommand.

**Tcl_DeleteCommand** deletes a command from a command interpreter.
Once the call completes, attempts to invoke *cmdName* in *interp* will
result in errors. If *cmdName* is not bound as a command in *interp* then
**Tcl_DeleteCommand** does nothing and returns -1; otherwise it returns
0. There are no restrictions on *cmdName*: it may refer to a built-in
command, an application-specific command, or a Tcl procedure. If
*name* contains any :: namespace qualifiers, the command is deleted
from the specified namespace.

Given a token returned by Tcl_CreateObjCommand,
Tcl_DeleteCommandFromToken deletes the command from a
command interpreter. It will delete a command even if that command
has been renamed. Once the call completes, attempts to invoke the
command in *interp* will result in errors. If the command corresponding to
token has already been deleted from *interp* then Tcl_DeleteCommand
does nothing and returns -1; otherwise it returns 0.

Tcl_GetCommandInfo checks to see whether its *cmdName* argument
exists as a command in interp. cmdName may include :: namespace qualifiers to identify a command in a particular namespace. If the command is not found, then it returns 0. Otherwise it places information about the command in the Tcl_CmdInfo structure pointed to by infoPtr and returns 1. A Tcl_CmdInfo structure has the following fields:

```c
typedef struct Tcl_CmdInfo {
    int isNativeObjectProc;
    Tcl_ObjCmdProc *objProc;
    ClientData objClientData;
    Tcl_CmdProc *proc;
    ClientData clientData;
    Tcl_CmdDeleteProc *deleteProc;
    ClientData deleteData;
    Tcl_Namespace *namespacePtr;
} Tcl_CmdInfo;
```

The isNativeObjectProc field has the value 1 if Tcl_CreateObjCommand was called to register the command; it is 0 if only Tcl_CreateCommand was called. It allows a program to determine whether it is faster to call objProc or proc: objProc is normally faster if isNativeObjectProc has the value 1. The fields objProc and objClientData have the same meaning as the proc and clientData arguments to Tcl_CreateObjCommand; they hold information about the object-based command procedure that the Tcl interpreter calls to implement the command. The fields proc and clientData hold information about the string-based command procedure that implements the command. If Tcl_CreateCommand was called for this command, this is the procedure passed to it; otherwise, this is a compatibility procedure registered by Tcl_CreateObjCommand that simply calls the command’s object-based procedure after converting its string arguments to Tcl objects. The field deleteData is the ClientData value to pass to deleteProc; it is normally the same as clientData but may be set independently using the Tcl_SetCommandInfo procedure. The field namespacePtr holds a pointer to the Tcl_Namespace that contains the command.
Tcl_GetCommandInfoFromToken is identical to Tcl_GetCommandInfo except that it uses a command token returned from Tcl_CreateObjCommand in place of the command name. If the token parameter is NULL, it returns 0; otherwise, it returns 1 and fills in the structure designated by infoPtr.

Tcl_SetCommandInfo is used to modify the procedures and ClientData values associated with a command. Its cmdName argument is the name of a command in interp. cmdName may include :: namespace qualifiers to identify a command in a particular namespace. If this command does not exist then Tcl_SetCommandInfo returns 0. Otherwise, it copies the information from *infoPtr to Tcl's internal structure for the command and returns 1.

Tcl_SetCommandInfoFromToken is identical to Tcl_SetCommandInfo except that it takes a command token as returned by Tcl_CreateObjCommand instead of the command name. If the token parameter is NULL, it returns 0. Otherwise, it copies the information from *infoPtr to Tcl's internal structure for the command and returns 1.

Note that Tcl_SetCommandInfo and Tcl_SetCommandInfoFromToken both allow the ClientData for a command's deletion procedure to be given a different value than the ClientData for its command procedure.

Note that neither Tcl_SetCommandInfo nor Tcl_SetCommandInfoFromToken will change a command's namespace. Use Tcl_Eval to call the rename command to do that.

Tcl_GetCommandName provides a mechanism for tracking commands that have been renamed. Given a token returned by Tcl_CreateObjCommand when the command was created, Tcl_GetCommandName returns the string name of the command. If the command has been renamed since it was created, then Tcl_GetCommandName returns the current name. This name does not include any :: namespace qualifiers. The command corresponding to token must not have been deleted. The string returned by
**Tcl_GetCommandName** is in dynamic memory owned by Tcl and is only guaranteed to retain its value as long as the command is not deleted or renamed; callers should copy the string if they need to keep it for a long time.

**Tcl_GetCommandFullName** produces the fully qualified name of a command from a command token. The name, including all namespace prefixes, is appended to the object specified by `objPtr`.

**Tcl_GetCommandFromObj** returns a token for the command specified by the name in a **Tcl_Obj**. The command name is resolved relative to the current namespace. Returns NULL if the command is not found.

**SEE ALSO**

**Tcl_CreateCommand**, **Tcl_ResetResult**, **Tcl_SetObjResult**

**KEYWORDS**

bind, command, create, delete, namespace, object

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996-1997 Sun Microsystems, Inc.
NAME
Tcl_SplitList, Tcl_Merge, Tcl_ScanElement,
Tcl_ConvertElement, Tcl_ScanCountedElement,
Tcl_ConvertCountedElement - manipulate Tcl lists

SYNOPSIS
#include <tcl.h>
int
Tcl_SplitList(interp, list, argcPtr, argvPtr)
char *
Tcl_Merge(argc, argv)
int
Tcl_ScanElement(src, flagsPtr)
int
Tcl_ScanCountedElement(src, length, flagsPtr)
int
Tcl_ConvertElement(src, dst, flags)
int
Tcl_ConvertCountedElement(src, length, dst, flags)

ARGUMENTS
DESCRIPTION
KEYWORDS

NAME
Tcl_SplitList, Tcl_Merge, Tcl_ScanElement, Tcl_ConvertElement,
Tcl_ScanCountedElement, Tcl_ConvertCountedElement - manipulate
Tcl lists

SYNOPSIS
#include <tcl.h>
int
**Tcl_SplitList** (*interp, list, argcPtr, argvPtr*)

- **interp** (out)
- **list** (in)
- **argcPtr** (out)
- **argvPtr** (out)

**Tcl_Merge** (*argc, argv*)

- **argc** (in)
- **argv** (in)

**Tcl_ScanElement** (*src, flagsPtr*)

- **src**
- **flagsPtr**

**Tcl_ScanCountedElement** (*src, length, flagsPtr*)

- **src**
- **length**
- **flagsPtr**

**Tcl_ConvertElement** (*src, dst, flags*)

- **src**
- **dst**
- **flags**

**Tcl_ConvertCountedElement** (*src, length, dst, flags*)

- **src**
- **length**
- **dst**
- **flags**

### ARGUMENTS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong></td>
<td>Interpreter to use for error reporting. If NULL, then no error message is left.</td>
</tr>
<tr>
<td><em>interp</em> (out)</td>
<td></td>
</tr>
<tr>
<td><strong>list</strong> (in)</td>
<td>Pointer to a string with proper list structure.</td>
</tr>
<tr>
<td><strong>argcPtr</strong> (out)</td>
<td>Filled in with number of elements in <em>list</em>.</td>
</tr>
<tr>
<td><strong>argvPtr</strong> (out)</td>
<td>*argvPtr will be filled in with the address of an array of pointers to the strings that are the extracted elements of <em>list</em>. There will be *argcPtr valid entries in the array, followed by a NULL entry.</td>
</tr>
<tr>
<td><strong>argc</strong> (in)</td>
<td>Number of elements in <em>argv</em>.</td>
</tr>
<tr>
<td><strong>argv</strong> (in)</td>
<td>Array of strings to merge.</td>
</tr>
</tbody>
</table>
These procedures may be used to disassemble and reassemble Tcl lists. **Tcl_SplitList** breaks a list up into its constituent elements, returning an array of pointers to the elements using `argcPtr` and `argvPtr`. While extracting the arguments, **Tcl_SplitList** obeys the usual rules for backslash substitutions and braces. The area of memory pointed to by
*argvPtr is dynamically allocated; in addition to the array of pointers, it also holds copies of all the list elements. It is the caller's responsibility to free up all of this storage. For example, suppose that you have called Tcl_SplitList with the following code:

```c
int argc, code;
char *string;
char **argv;
...
code = Tcl_SplitList(interp, string, &argc, &argv);
```

Then you should eventually free the storage with a call like the following:

```c
Tcl_Free((char *) argv);
```

Tcl_SplitList normally returns TCL_OK, which means the list was successfully parsed. If there was a syntax error in list, then TCL_ERROR is returned and the interpreter's result will point to an error message describing the problem (if interp was not NULL). If TCL_ERROR is returned then no memory is allocated and *argvPtr is not modified.

Tcl_Merge is the inverse of Tcl_SplitList: it takes a collection of strings given by argc and argv and generates a result string that has proper list structure. This means that commands like index may be used to extract the original elements again. In addition, if the result of Tcl_Merge is passed to Tcl_Eval, it will be parsed into argc words whose values will be the same as the argv strings passed to Tcl_Merge. Tcl_Merge will modify the list elements with braces and/or backslashes in order to produce proper Tcl list structure. The result string is dynamically allocated using Tcl_Alloc; the caller must eventually release the space using Tcl_Free.
If the result of `Tcl_Merge` is passed to `Tcl_SplitList`, the elements returned by `Tcl_SplitList` will be identical to those passed into `Tcl_Merge`. However, the converse is not true: if `Tcl_SplitList` is passed a given string, and the resulting `argv` and `argc` are passed to `Tcl_Merge`, the resulting string may not be the same as the original string passed to `Tcl_SplitList`. This is because `Tcl_Merge` may use backslashes and braces differently than the original string.

`Tcl_ScanElement` and `Tcl_ConvertElement` are the procedures that do all of the real work of `Tcl_Merge`. `Tcl_ScanElement` scans its `src` argument and determines how to use backslashes and braces when converting it to a list element. It returns an overestimate of the number of characters required to represent `src` as a list element, and it stores information in `*flagsPtr` that is needed by `Tcl_ConvertElement`.

`Tcl_ConvertElement` is a companion procedure to `Tcl_ScanElement`. It does the actual work of converting a string to a list element. Its `flags` argument must be the same as the value returned by `Tcl_ScanElement`. `Tcl_ConvertElement` writes a proper list element to memory starting at `*dst` and returns a count of the total number of characters written, which will be no more than the result returned by `Tcl_ScanElement`. `Tcl_ConvertElement` writes out only the actual list element without any leading or trailing spaces: it is up to the caller to include spaces between adjacent list elements.

`Tcl_ConvertElement` uses one of two different approaches to handle the special characters in `src`. Wherever possible, it handles special characters by surrounding the string with braces. This produces clean-looking output, but cannot be used in some situations, such as when `src` contains unmatched braces. In these situations, `Tcl_ConvertElement` handles special characters by generating backslash sequences for them. The caller may insist on the second approach by OR-ing the flag value returned by `Tcl_ScanElement` with `TCL_DONT_USE_BRACES`. Although this will produce an uglier result, it is useful in some special situations, such as when `Tcl_ConvertElement` is being used to generate a portion of an argument for a Tcl command. In this case, surrounding `src` with curly braces would cause the command not to be parsed correctly.
By default, **Tcl_ConvertElement** will use quoting in its output to be sure the first character of an element is not the hash character ("#".) This is to be sure the first element of any list passed to **eval** is not mis-parsed as the beginning of a comment. When a list element is not the first element of a list, this quoting is not necessary. When the caller can be sure that the element is not the first element of a list, it can disable quoting of the leading hash character by OR-ing the flag value returned by **Tcl_ScanElement** with **TCL_DONT_QUOTE_HASH**.

**Tcl_ScanCountedElement** and **Tcl_ConvertCountedElement** are the same as **Tcl_ScanElement** and **Tcl_ConvertElement**, except the length of string *src* is specified by the *length* argument, and the string may contain embedded nulls.

**KEYWORDS**

backslash, convert, element, list, merge, split, strings

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tcl_GetCwd, Tcl_Chdir - manipulate the current working directory

SYNOPSIS
#include <tcl.h>
char *
Tcl_GetCwd(interp, bufferPtr)
int
Tcl_Chdir(path)

ARGUMENTS

DESCRIPTION

KEYWORDS

NAME
Tcl_GetCwd, Tcl_Chdir - manipulate the current working directory

SYNOPSIS
#include <tcl.h>
char *
Tcl_GetCwd(interp, bufferPtr)
int
Tcl_Chdir(path)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter in which to report an error, if any.

Tcl_DString *bufferPtr (in/out)
This dynamic string is used to store the current
These procedures may be used to manipulate the current working directory for the application. They provide C-level access to the same functionality as the Tcl `pwd` command.

`Tcl_GetCwd` returns a pointer to a string specifying the current directory, or NULL if the current directory could not be determined. If NULL is returned, an error message is left in the `interp's` result. Storage for the result string is allocated in bufferPtr; the caller must call `Tcl_DStringFree()` when the result is no longer needed. The format of the path is UTF-8.

`Tcl_Chdir` changes the applications current working directory to the value specified in `path`. The format of the passed in string must be UTF-8. The function returns -1 on error or 0 on success.
NAME

Tcl_CommandComplete - Check for unmatched braces in a Tcl command

SYNOPSIS

#include <tcl.h>

int
Tcl_CommandComplete(cmd)

ARGUMENTS

const char *cmd (in) Command string to test for completeness.

DESCRIPTION

Tcl_CommandComplete takes a Tcl command string as argument and determines whether it contains one or more complete commands (i.e. there are no unclosed quotes, braces, brackets, or variable references). If the command string is complete then it returns 1; otherwise it returns 0.

KEYWORDS

complete command, partial command

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tcl_GetEncoding, Tcl_FreeEncoding,
Tcl_GetEncodingFromObj, Tcl.getExternalToUtfDString,
Tcl_getEncToUtf, Tcl_UtfToExternalDString,
Tcl_UtfToExternal, Tcl_WinTCharToUtf, Tcl_WinUtfToTChar,
Tcl_GetEncodingName, Tcl_SetSystemEncoding,
Tcl_GetEncodingNameFromEnvironment,
Tcl_GetEncodingNames, Tcl_CreateEncoding,
Tcl_GetEncodingSearchPath, Tcl_SetEncodingSearchPath,
Tcl_GetDefaultEncodingDir, Tcl_SetDefaultEncodingDir -
procedures for creating and using encodings

SYNOPSIS
#include <tcl.h>
Tcl_Encoding
Tcl_GetEncoding(interp, name)
void
Tcl_FreeEncoding(encoding)
int
Tcl_GetEncodingFromObj(interp, objPtr, encodingPtr)
char *
Tcl_ExternalToUtfDString(encoding, src, srcLen, dstPtr)
char *
Tcl_UtfToExternalDString(encoding, src, srcLen, dstPtr)
int
Tcl_ExternalToUtf(interp, encoding, src, srcLen, flags, statePtr,
dst, dstLen, srcReadPtr, dstWrotePtr, dstCharsPtr)
int
Tcl_UtfToExternal(interp, encoding, src, srcLen, flags, statePtr,
dst, dstLen, srcReadPtr, dstWrotePtr, dstCharsPtr)
char *
Tcl_WinTCharToUtf(tsrc, srclen, dstPtr)
TCHAR *
Tcl_WinUtfToTChar(src, srclen, dstPtr)
const char *
Tcl_GetEncodingName(encoding)
int
Tcl_SetSystemEncoding(interp, name)
const char *
Tcl_GetEncodingNameFromEnvironment(bufPtr)
void
Tcl_GetEncodingNames(interp)
Tcl_Encoding
Tcl_CreateEncoding(typePtr)
Tcl_Obj *
Tcl_GetEncodingSearchPath()
int
Tcl_SetEncodingSearchPath(searchPath)
const char *
Tcl_GetDefaultEncodingDir(void)
void
Tcl_SetDefaultEncodingDir(path)

ARGUMENTS
INTRODUCTION
DESCRIPTION
TCL_OK
TCL_CONVERT_NOSPACE
TCL_CONVERT_MULTIBYTE
TCL_CONVERT_SYNTAX
TCL_CONVERT_UNKNOWN
ENCODING FILES
[1] S
[2] D
[3] M
[4] E
KEYWORDS
NAME

Tcl_GetEncoding, Tcl_FreeEncoding, Tcl_GetEncodingFromObj,
Tcl_ExternalToUtfDString, Tcl_ExternalToUtf, Tcl_UtfToExternalDString,
Tcl_UtfToExternal, Tcl_WinTCharToUtf, Tcl_WinUtfToTChar,
Tcl_GetEncodingName, Tcl_SetSystemEncoding,
Tcl_GetEncodingNameFromEnvironment, Tcl_GetEncodingNames,
Tcl_CreateEncoding, Tcl_GetEncodingSearchPath,
Tcl_SetEncodingSearchPath, Tcl_GetDefaultEncodingDir,
Tcl_SetDefaultEncodingDir - procedures for creating and using encodings

SYNOPSIS

#include <tcl.h>
Tcl_Encoding
Tcl_GetEncoding(interp, name)
void
Tcl_FreeEncoding(encoding)
int
Tcl_GetEncodingFromObj(interp, objPtr, encodingPtr)
char *
Tcl_ExternalToUtfDString(encoding, src, srcLen, dstPtr)
char *
Tcl_UtfToExternalDString(encoding, src, srcLen, dstPtr)
int
Tcl_ExternalToUtf(interp, encoding, src, srcLen, flags, statePtr,
dst, dstLen, srcReadPtr, dstWrotePtr, dstCharsPtr)
int
Tcl_UtfToExternal(interp, encoding, src, srcLen, flags, statePtr,
dst, dstLen, srcReadPtr, dstWrotePtr, dstCharsPtr)
char *
Tcl_WinTCharToUtf(tsrc, srcLen, dstPtr)
TCHAR *
Tcl_WinUtfToTChar(src, srcLen, dstPtr)
const char *
Tcl_GetEncodingName(encoding)
int
Tcl_SetSystemEncoding(interp, name)
const char *
Tcl_GetEncodingNameFromEnvironment(bufPtr)
void
Tcl_GetEncodingNames(interp)
Tcl_Encoding
Tcl_CreateEncoding(typePtr)
Tcl_Obj *
Tcl_GetEncodingSearchPath()
int
Tcl_SetEncodingSearchPath(searchPath)
const char *
Tcl_GetDefaultEncodingDir(void)
void
Tcl_SetDefaultEncodingDir(path)

<p>| ARGUMENTS |
|-----------------------------------------------|--------------------------------------------------|
| *<em>Tcl_Interp <em>interp (in)</em></em>                    | Interpreter to use for error reporting, or NULL if no error reporting is desired. |
| *<em>const char <em>name (in)</em></em>                      | Name of encoding to load.                        |
| <strong>Tcl_Encoding encoding (in)</strong>                 | The encoding to query, free, or use for converting text. If <em>encoding</em> is NULL, the current system encoding is used. |
| *<em>Tcl_Obj <em>objPtr (in)</em></em>                       | Name of encoding to get token for.               |
| *<em>Tcl_Encoding <em>encodingPtr (out)</em></em>            | Points to storage where encoding token is to be written. |</p>
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const char *src (in)</td>
<td>For the <code>Tcl_ExternalToUtf</code> functions, an array of bytes in the specified encoding that are to be converted to UTF-8. For the <code>Tcl_UtfToExternal</code> and <code>Tcl_WinUtfToTChar</code> functions, an array of UTF-8 characters to be converted to the specified encoding.</td>
</tr>
<tr>
<td>const TCHAR *tsrc (in)</td>
<td>An array of Windows TCHAR characters to convert to UTF-8.</td>
</tr>
<tr>
<td>int srcLen (in)</td>
<td>Length of src or tsrc in bytes. If the length is negative, the encoding-specific length of the string is used.</td>
</tr>
<tr>
<td>Tcl_DString *dstPtr (out)</td>
<td>Pointer to an uninitialized or free <code>Tcl_DString</code> in which the converted result will be stored.</td>
</tr>
<tr>
<td>int flags (in)</td>
<td>Various flag bits OR-ed together. <code>TCL_ENCODING_START</code> signifies that the source buffer is the first block in a (potentially multi-block) input stream, telling the conversion routine to reset to an initial state and perform any initialization</td>
</tr>
</tbody>
</table>
that needs to occur before the first byte is converted.

**TCL_ENCODING_END** signifies that the source buffer is the last block in a (potentially multi-block) input stream, telling the conversion routine to perform any finalization that needs to occur after the last byte is converted and then to reset to an initial state.

**TCL_ENCODING_STOPONERROR** signifies that the conversion routine should return immediately upon reading a source character that does not exist in the target encoding; otherwise a default fallback character will automatically be substituted.

---

**Tcl_EncodingState **:*statePtr* (in/out)  
Used when converting a (generally long or indefinite length) byte stream in a piece-by-piece fashion. The conversion routine stores its current state in *statePtr* after *src* (the buffer containing the current piece) has been converted; that state information must be passed back when converting the next piece.
of the stream so the conversion routine knows what state it was in when it left off at the end of the last piece. May be NULL, in which case the value specified for flags is ignored and the source buffer is assumed to contain the complete string to convert.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char *dst (out)</code></td>
<td>Buffer in which the converted result will be stored. No more than <code>dstLen</code> bytes will be stored in <code>dst</code>.</td>
</tr>
<tr>
<td><code>int dstLen (in)</code></td>
<td>The maximum length of the output buffer <code>dst</code> in bytes.</td>
</tr>
<tr>
<td><code>int *srcReadPtr (out)</code></td>
<td>Filled with the number of bytes from <code>src</code> that were actually converted. This may be less than the original source length if there was a problem converting some source characters. May be NULL.</td>
</tr>
<tr>
<td><code>int *dstWrotePtr (out)</code></td>
<td>Filled with the number of bytes that were actually stored in the output buffer as a result of the conversion. May be NULL.</td>
</tr>
<tr>
<td>*<em>int <em>dstCharsPtr</em></em> (out)</td>
<td>Filled with the number of characters that correspond to the number of bytes stored in the output buffer. May be NULL.</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>*<em>Tcl_DString <em>bufPtr</em></em> (out)</td>
<td>Storage for the prescribed system encoding name.</td>
</tr>
<tr>
<td>*<em>const Tcl_EncodingType <em>typePtr</em></em> (in)</td>
<td>Structure that defines a new type of encoding.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>searchPath</em></em> (in)</td>
<td>List of filesystem directories in which to search for encoding data files.</td>
</tr>
<tr>
<td>*<em>const char <em>path</em></em> (in)</td>
<td>A path to the location of the encoding file.</td>
</tr>
</tbody>
</table>

**INTRODUCTION**

These routines convert between Tcl's internal character representation, UTF-8, and character representations used by various operating systems or file systems, such as Unicode, ASCII, or Shift-JIS. When operating on strings, such as such as obtaining the names of files or displaying characters using international fonts, the strings must be translated into one or possibly multiple formats that the various system calls can use. For instance, on a Japanese Unix workstation, a user might obtain a filename represented in the EUC-JP file encoding and then translate the characters to the jisx0208 font encoding in order to display the filename in a Tk widget. The purpose of the encoding package is to help bridge the translation gap. UTF-8 provides an intermediate staging ground for all the various encodings. In the example above, text would be translated into UTF-8 from whatever file encoding the operating system is using. Then it would be translated...
from UTF-8 into whatever font encoding the display routines require.

Some basic encodings are compiled into Tcl. Others can be defined by the user or dynamically loaded from encoding files in a platform-independent manner.

**DESCRIPTION**

*Tcl_GetEncoding* finds an encoding given its *name*. The name may refer to a built-in Tcl encoding, a user-defined encoding registered by calling *Tcl_CreateEncoding*, or a dynamically-loadable encoding file. The return value is a token that represents the encoding and can be used in subsequent calls to procedures such as *Tcl_GetEncodingName*, *Tcl_FreeEncoding*, and *Tcl_UtfToExternal*. If the name did not refer to any known or loadable encoding, NULL is returned and an error message is returned in *interp*.

The encoding package maintains a database of all encodings currently in use. The first time *name* is seen, *Tcl_GetEncoding* returns an encoding with a reference count of 1. If the same *name* is requested further times, then the reference count for that encoding is incremented without the overhead of allocating a new encoding and all its associated data structures.

When an *encoding* is no longer needed, *Tcl_FreeEncoding* should be called to release it. When an *encoding* is no longer in use anywhere (i.e., it has been freed as many times as it has been gotten) *Tcl_FreeEncoding* will release all storage the encoding was using and delete it from the database.

*Tcl_GetEncodingFromObj* treats the string representation of *objPtr* as an encoding name, and finds an encoding with that name, just as *Tcl_GetEncoding* does. When an encoding is found, it is cached within the *objPtr* value for future reference, the *Tcl_Encoding* token is written to the storage pointed to by *encodingPtr*, and the value TCL_OK is returned. If no such encoding is found, the value TCL_ERROR is returned, and no writing to *encodingPtr* takes place. Just as with *Tcl_GetEncoding*, the caller should call *Tcl_FreeEncoding* on the
resulting encoding token when that token will no longer be used.

**Tcl.getExternalToUtfDString** converts a source buffer src from the specified *encoding* into UTF-8. The converted bytes are stored in dstPtr, which is then null-terminated. The caller should eventually call **Tcl.DStringFree** to free any information stored in dstPtr. When converting, if any of the characters in the source buffer cannot be represented in the target encoding, a default fallback character will be used. The return value is a pointer to the value stored in the DString.

**Tcl.getExternalToUtf** converts a source buffer src from the specified *encoding* into UTF-8. Up to srcLen bytes are converted from the source buffer and up to dstLen converted bytes are stored in dst. In all cases, *srcReadPtr* is filled with the number of bytes that were successfully converted from src and *dstWrotePtr* is filled with the corresponding number of bytes that were stored in dst. The return value is one of the following:

**TCL_OK**
All bytes of src were converted.

**TCL_CONVERT_NOSPACE**
The destination buffer was not large enough for all of the converted data; as many characters as could fit were converted though.

**TCL_CONVERT_MULTIBYTE**
The last few bytes in the source buffer were the beginning of a multibyte sequence, but more bytes were needed to complete this sequence. A subsequent call to the conversion routine should pass a buffer containing the unconverted bytes that remained in src plus some further bytes from the source stream to properly convert the formerly split-up multibyte sequence.

**TCL_CONVERT_SYNTAX**
The source buffer contained an invalid character sequence. This may occur if the input stream has been damaged or if the input encoding method was misidentified.
TCL_CONVERT_UNKNOWN
The source buffer contained a character that could not be represented in the target encoding and TCL_ENCODING_STOPONERROR was specified.

TclUtfToExternalDString converts a source buffer src from UTF-8 into the specified encoding. The converted bytes are stored in dstPtr, which is then terminated with the appropriate encoding-specific null. The caller should eventually call Tcl_DStringFree to free any information stored in dstPtr. When converting, if any of the characters in the source buffer cannot be represented in the target encoding, a default fallback character will be used. The return value is a pointer to the value stored in the DString.

TclUtfToExternal converts a source buffer src from UTF-8 into the specified encoding. Up to srcLen bytes are converted from the source buffer and up to dstLen converted bytes are stored in dst. In all cases, *srcReadPtr is filled with the number of bytes that were successfully converted from src and *dstWrotePtr is filled with the corresponding number of bytes that were stored in dst. The return values are the same as the return values for Tcl_ExternalToUtf.

TclWinUtfToTChar and TclWinTCharToUtf are Windows-only convenience functions for converting between UTF-8 and Windows strings. On Windows 95 (as with the Unix operating system), all strings exchanged between Tcl and the operating system are “char” based. On Windows NT, some strings exchanged between Tcl and the operating system are “char” oriented while others are in Unicode. By convention, in Windows a TCHAR is a character in the ANSI code page on Windows 95 and a Unicode character on Windows NT.

If you planned to use the same “char” based interfaces on both Windows 95 and Windows NT, you could use TclUtfToExternal and TclExternalToUtf (or their TclDString equivalents) with an encoding of NULL (the current system encoding). On the other hand, if you planned to use the Unicode interface when running on Windows NT and the “char” interfaces when running on Windows 95, you would have to perform the following type of test over and over in your program (as
represented in pseudo-code):

```c
if (running_NT) {
    encoding <- Tcl_GetEncoding("unicode");
    nativeBuffer <- Tcl_UtfToExternal(encoding, utfB
    Tcl_FreeEncoding(encoding);
} else {
    nativeBuffer <- Tcl_UtfToExternal(NULL, utfBuffe
```

Tcl_WinUtfToTChar and Tcl_WinTCharToUtf automatically handle this
test and use the proper encoding based on the current operating
system. Tcl_WinUtfToTChar returns a pointer to a TCHAR string, and
Tcl_WinTCharToUtf expects a TCHAR string pointer as the src string.
Otherwise, these functions behave identically to
Tcl_UtfToExternalDString and Tcl_ExternalToUtfDString.

Tcl_GetEncodingName is roughly the inverse of Tcl_GetEncoding. Given an encoding, the return value is the name argument that was
used to create the encoding. The string returned by
Tcl_GetEncodingName is only guaranteed to persist until the encoding is deleted. The caller must not modify this string.

Tcl_SetSystemEncoding sets the default encoding that should be
used whenever the user passes a NULL value for the encoding argument to any of the other encoding functions. If name is NULL, the
system encoding is reset to the default system encoding, binary. If the
name did not refer to any known or loadable encoding, TCL_ERROR is
returned and an error message is left in interp. Otherwise, this
procedure increments the reference count of the new system encoding,
decrements the reference count of the old system encoding, and
returns TCL_OK.

Tcl_GetEncodingNameFromEnvironment provides a means for the
Tcl library to report the encoding name it believes to be the correct one
to use as the system encoding, based on system calls and examination of the environment suitable for the platform. It accepts *bufPtr*, a pointer to an uninitialized or freed *Tcl_DString* and writes the encoding name to it. The *Tcl_DStringValue* is returned.

**Tcl_GetEncodingNames** sets the *interp* result to a list consisting of the names of all the encodings that are currently defined or can be dynamically loaded, searching the encoding path specified by **Tcl_SetDefaultEncodingDir**. This procedure does not ensure that the dynamically-loadable encoding files contain valid data, but merely that they exist.

**Tcl_CreateEncoding** defines a new encoding and registers the C procedures that are called back to convert between the encoding and UTF-8. Encodings created by **Tcl_CreateEncoding** are thereafter visible in the database used by **Tcl_GetEncoding**. Just as with the **Tcl_GetEncoding** procedure, the return value is a token that represents the encoding and can be used in subsequent calls to other encoding functions. **Tcl_CreateEncoding** returns an encoding with a reference count of 1. If an encoding with the specified *name* already exists, then its entry in the database is replaced with the new encoding; the token for the old encoding will remain valid and continue to behave as before, but users of the new token will now call the new encoding procedures.

The *typePtr* argument to **Tcl_CreateEncoding** contains information about the name of the encoding and the procedures that will be called to convert between this encoding and UTF-8. It is defined as follows:

```c
typedef struct Tcl_EncodingType {
    const char *encodingName;
    Tcl_EncodingConvertProc *toUtfProc;
    Tcl_EncodingConvertProc *fromUtfProc;
    Tcl_EncodingFreeProc *freeProc;
    ClientData clientData;
    int nullSize;
} Tcl_EncodingType;
```
The `encodingName` provides a string name for the encoding, by which it can be referred in other procedures such as `Tcl_GetEncoding`. The `toUtfProc` refers to a callback procedure to invoke to convert text from this encoding into UTF-8. The `fromUtfProc` refers to a callback procedure to invoke to convert text from UTF-8 into this encoding. The `freeProc` refers to a callback procedure to invoke when this encoding is deleted. The `freeProc` field may be NULL. The `clientData` contains an arbitrary one-word value passed to `toUtfProc`, `fromUtfProc`, and `freeProc` whenever they are called. Typically, this is a pointer to a data structure containing encoding-specific information that can be used by the callback procedures. For instance, two very similar encodings such as `ascii` and `macRoman` may use the same callback procedure, but use different values of `clientData` to control its behavior. The `nullSize` specifies the number of zero bytes that signify end-of-string in this encoding. It must be 1 (for single-byte or multi-byte encodings like ASCII or Shift-JIS) or 2 (for double-byte encodings like Unicode). Constant-sized encodings with 3 or more bytes per character (such as CNS11643) are not accepted.

The callback procedures `toUtfProc` and `fromUtfProc` should match the type `Tcl_EncodingConvertProc`:

```c
typedef int Tcl_EncodingConvertProc(
    ClientData clientData,
    const char *src,
    int srcLen,
    int flags,
    Tcl_EncodingState *statePtr,
    char *dst,
    int dstLen,
    int *srcReadPtr,
    int *dstWrotePtr,
    int *dstCharsPtr);
```
The `toUtfProc` and `fromUtfProc` procedures are called by the `Tcl_ExternalToUtf` or `Tcl_UtfToExternal` family of functions to perform the actual conversion. The `clientData` parameter to these procedures is the same as the `clientData` field specified to `Tcl_CreateEncoding` when the encoding was created. The remaining arguments to the callback procedures are the same as the arguments, documented at the top, to `Tcl_ExternalToUtf` or `Tcl_UtfToExternal`, with the following exceptions. If the `srcLen` argument to one of those high-level functions is negative, the value passed to the callback procedure will be the appropriate encoding-specific string length of `src`. If any of the `srcReadPtr`, `dstWrotePtr`, or `dstCharsPtr` arguments to one of the high-level functions is NULL, the corresponding value passed to the callback procedure will be a non-NULL location.

The callback procedure `freeProc`, if non-NULL, should match the type `Tcl_EncodingFreeProc`:

```c
typedef void Tcl_EncodingFreeProc(
    ClientData clientData);
```

This `freeProc` function is called when the encoding is deleted. The `clientData` parameter is the same as the `clientData` field specified to `Tcl_CreateEncoding` when the encoding was created.

`Tcl_GetEncodingSearchPath` and `Tcl_SetEncodingSearchPath` are called to access and set the list of filesystem directories searched for encoding data files.

The value returned by `Tcl_GetEncodingSearchPath` is the value stored by the last successful call to `Tcl_SetEncodingSearchPath`. If no calls to `Tcl_SetEncodingSearchPath` have occurred, Tcl will compute an initial value based on the environment. There is one encoding search path for the entire process, shared by all threads in the process.

`Tcl_SetEncodingSearchPath` stores `searchPath` and returns `TCL_OK`, unless `searchPath` is not a valid Tcl list, which causes `TCL_ERROR` to
be returned. The elements of searchPath are not verified as existing readable filesystem directories. When searching for encoding data files takes place, and non-existent or non-readable filesystem directories on the searchPath are silently ignored.

Tcl_GetDefaultEncodingDir and Tcl_SetDefaultEncodingDir are obsolete interfaces best replaced with calls to Tcl_GetEncodingSearchPath and Tcl_SetEncodingSearchPath. They are called to access and set the first element of the searchPath list. Since Tcl searches searchPath for encoding data files in list order, these routines establish the “default” directory in which to find encoding data files.

ENCODING FILES

Space would prohibit precompiling into Tcl every possible encoding algorithm, so many encodings are stored on disk as dynamically-loadable encoding files. This behavior also allows the user to create additional encoding files that can be loaded using the same mechanism. These encoding files contain information about the tables and/or escape sequences used to map between an external encoding and Unicode. The external encoding may consist of single-byte, multi-byte, or double-byte characters.

Each dynamically-loadable encoding is represented as a text file. The initial line of the file, beginning with a “#” symbol, is a comment that provides a human-readable description of the file. The next line identifies the type of encoding file. It can be one of the following letters:

[1] S
A single-byte encoding, where one character is always one byte long in the encoding. An example is iso8859-1, used by many European languages.

[2] D
A double-byte encoding, where one character is always two bytes long in the encoding. An example is big5, used for Chinese text.
[3] M
A multi-byte encoding, where one character may be either one or two bytes long. Certain bytes are lead bytes, indicating that another byte must follow and that together the two bytes represent one character. Other bytes are not lead bytes and represent themselves. An example is shiftjis, used by many Japanese computers.

[4] E
An escape-sequence encoding, specifying that certain sequences of bytes do not represent characters, but commands that describe how following bytes should be interpreted.

The rest of the lines in the file depend on the type.

Cases [1], [2], and [3] are collectively referred to as table-based encoding files. The lines in a table-based encoding file are in the same format as this example taken from the shiftjis encoding (this is not the complete file):

```
# Encoding file: shiftjis, multi-byte
M
003F 0 40
00
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
```

```
0010001100120013001400150016001700180019001A001B001C
0020002100220023002400250026002700280029002A002B002C
0030003100320033003400350036003700380039003A003B003C
0040004100420043004400450046004700480049004A004B004C
0050005100520053005400550056005700580059005A005B005C
0060006100620063006400650066006700680069006A006B006C
0070007100720073007400750076007700780079007A007B007C
0080000000000000000000000000000000000000000000000000000000000000
0000FFFF61FF62FF63FF64FF65FF66FF67FF68FF69FF6AFF6BFF6C
FF70FF71FF72FF73FF74FF75FF76FF77FF78FF79FF7AFF7BFF7C
FF80FF81FF82FF83FF84FF85FF86FF87FF88FF89FF8AFF8BFF8C
```
The third line of the file is three numbers. The first number is the fallback character (in base 16) to use when converting from UTF-8 to this encoding. The second number is a 1 if this file represents the encoding for a symbol font, or 0 otherwise. The last number (in base 10) is how many pages of data follow.

Subsequent lines in the example above are pages that describe how to map from the encoding into 2-byte Unicode. The first line in a page identifies the page number. Following it are 256 double-byte numbers, arranged as 16 rows of 16 numbers. Given a character in the encoding, the high byte of that character is used to select which page, and the low byte of that character is used as an index to select one of the double-byte numbers in that page - the value obtained being the corresponding Unicode character. By examination of the example above, one can see that the characters 0x7E and 0x8163 in *shiftjis* map to 203E and 2026.
in Unicode, respectively.

Following the first page will be all the other pages, each in the same format as the first: one number identifying the page followed by 256 double-byte Unicode characters. If a character in the encoding maps to the Unicode character 0000, it means that the character does not actually exist. If all characters on a page would map to 0000, that page can be omitted.

Case [4] is the escape-sequence encoding file. The lines in an this type of file are in the same format as this example taken from the iso2022-jp encoding:

```plaintext
# Encoding file: iso2022-jp, escape-driven
E
init  {}
final {}
iso8859-1  \x1b(B
jis0201  \x1b(J
jis0208  \x1b@$
jis0208  \x1b$B
jis0212  \x1b$(D
gb2312  \x1b$A
ksc5601  \x1b$(C
```

In the file, the first column represents an option and the second column is the associated value. `init` is a string to emit or expect before the first character is converted, while `final` is a string to emit or expect after the last character. All other options are names of table-based encodings; the associated value is the escape-sequence that marks that encoding. Tcl syntax is used for the values; in the above example, for instance, “{}” represents the empty string and “\x1b” represents character 27.

When `Tcl_GetEncoding` encounters an encoding `name` that has not been loaded, it attempts to load an encoding file called `name.enc` from the `encoding` subdirectory of each directory that Tcl searches for its
script library. If the encoding file exists, but is malformed, an error message will be left in *interp*.

**KEYWORDS**

utf, encoding, convert

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997-1998 Sun Microsystems, Inc.
NAME
Tcl_CommandTraceInfo, Tcl_TraceCommand, Tcl_UntraceCommand - monitor renames and deletes of a command

SYNOPSIS
#include <tcl.h>
ClientData
Tcl_CommandTraceInfo(interp, cmdName, flags, proc, prevClientData)
int
Tcl_TraceCommand(interp, cmdName, flags, proc, clientData)
void
Tcl_UntraceCommand(interp, cmdName, flags, proc, clientData)

ARGUMENTS

DESCRIPTION
TCL_TRACE_RENAME
TCL_TRACE_DELETE

CALLING COMMANDS DURING TRACES

MULTIPLE TRACES

TCL_TRACE_DESTROYED FLAG
TCL_INTERP_DESTROYED

BUGS
KEYWORDS

NAME
Tcl_CommandTraceInfo, Tcl_TraceCommand, Tcl_UntraceCommand - monitor renames and deletes of a command

SYNOPSIS
#include <tcl.h>

ClientData

Tcl_CommandTraceInfo(interp, cmdName, flags, proc, prevClientData) int

Tcl_TraceCommand(interp, cmdName, flags, proc, clientData) void

Tcl_UntraceCommand(interp, cmdName, flags, proc, clientData)

### ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_Interp *interp (in)</td>
<td>Interpreter containing the command.</td>
</tr>
<tr>
<td>const char *cmdName (in)</td>
<td>Name of command.</td>
</tr>
<tr>
<td>int flags (in)</td>
<td>OR'ed collection of the values TCL_TRACE_RENAME and TCL_TRACE_DELETE.</td>
</tr>
<tr>
<td>Tcl_CommandTraceProc *proc (in)</td>
<td>Procedure to call when specified operations occur to cmdName.</td>
</tr>
<tr>
<td>ClientData clientData (in)</td>
<td>Arbitrary argument to pass to proc.</td>
</tr>
<tr>
<td>ClientData prevClientData (in)</td>
<td>If non-NULL, gives last value returned by Tcl_CommandTraceInfo, so this call will return information about next trace. If NULL, this call will return information about first trace.</td>
</tr>
</tbody>
</table>
**DESCRIPTION**

**Tcl_TraceCommand** allows a C procedure to monitor operations performed on a Tcl command, so that the C procedure is invoked whenever the command is renamed or deleted. If the trace is created successfully then **Tcl_TraceCommand** returns **TCL_OK**. If an error occurred (e.g. **cmdName** specifies a non-existent command) then **TCL_ERROR** is returned and an error message is left in the interpreter's result.

The **flags** argument to **Tcl_TraceCommand** indicates when the trace procedure is to be invoked. It consists of an OR'ed combination of any of the following values:

**TCL_TRACE_RENAME**
Invoke **proc** whenever the command is renamed.

**TCL_TRACE_DELETE**
Invoke **proc** when the command is deleted.

Whenever one of the specified operations occurs to the command, **proc** will be invoked. It should have arguments and result that match the type **Tcl_CommandTraceProc**:

```c
typedef void Tcl_CommandTraceProc(
    ClientData clientData,
    Tcl_Interp *interp,
    const char *oldName,
    const char *newName,
    int flags);
```

The **clientData** and **interp** parameters will have the same values as those passed to **Tcl_TraceCommand** when the trace was created. **ClientData** typically points to an application-specific data structure that describes what to do when **proc** is invoked. **OldName** gives the name of the command being renamed, and **newName** gives the name that the
command is being renamed to (or an empty string or NULL when the command is being deleted.) *Flags* is an OR'ed combination of bits potentially providing several pieces of information. One of the bits **TCL_TRACE_RENAME** and **TCL_TRACE_DELETE** will be set in *flags* to indicate which operation is being performed on the command. The bit **TCL_TRACE_DESTROYED** will be set in *flags* if the trace is about to be destroyed; this information may be useful to *proc* so that it can clean up its own internal data structures (see the section **TCL_TRACE_DESTROYED** below for more details). Lastly, the bit **TCL_INTERP_DESTROYED** will be set if the entire interpreter is being destroyed. When this bit is set, *proc* must be especially careful in the things it does (see the section **TCL_INTERP_DESTROYED** below).

**Tcl_UntraceCommand** may be used to remove a trace. If the command specified by *interp*, *cmdName*, and *flags* has a trace set with *flags*, *proc*, and *clientData*, then the corresponding trace is removed. If no such trace exists, then the call to **Tcl_UntraceCommand** has no effect. The same bits are valid for *flags* as for calls to **Tcl_TraceCommand**.

**Tcl_CommandTraceInfo** may be used to retrieve information about traces set on a given command. The return value from **Tcl_CommandTraceInfo** is the *clientData* associated with a particular trace. The trace must be on the command specified by the *interp*, *cmdName*, and *flags* arguments (note that currently the flags are ignored; *flags* should be set to 0 for future compatibility) and its trace procedure must the same as the *proc* argument. If the *prevClientData* argument is NULL then the return value corresponds to the first (most recently created) matching trace, or NULL if there are no matching traces. If the *prevClientData* argument is not NULL, then it should be the return value from a previous call to **Tcl_CommandTraceInfo**. In this case, the new return value will correspond to the next matching trace after the one whose *clientData* matches *prevClientData*, or NULL if no trace matches *prevClientData* or if there are no more matching traces after it. This mechanism makes it possible to step through all of the traces for a given command that have the same *proc*. 
CALLING COMMANDS DURING TRACES

During rename traces, the command being renamed is visible with both names simultaneously, and the command still exists during delete traces (if TCL_INTERP_DESTROYED is not set). However, there is no mechanism for signaling that an error occurred in a trace procedure, so great care should be taken that errors do not get silently lost.

MULTIPLE TRACES

It is possible for multiple traces to exist on the same command. When this happens, all of the trace procedures will be invoked on each access, in order from most-recently-created to least-recently-created. Attempts to delete the command during a delete trace will fail silently, since the command is already scheduled for deletion anyway. If the command being renamed is renamed by one of its rename traces, that renaming takes precedence over the one that triggered the trace and the collection of traces will not be reexecuted; if several traces rename the command, the last renaming takes precedence.

TCL_TRACE_DESTROYED FLAG

In a delete callback to proc, the TCL_TRACE_DESTROYED bit is set in flags.

TCL_INTERP_DESTROYED

When an interpreter is destroyed, unset traces are called for all of its commands. The TCL_INTERP_DESTROYED bit will be set in the flags argument passed to the trace procedures. Trace procedures must be extremely careful in what they do if the TCL_INTERP_DESTROYED bit is set. It is not safe for the procedures to invoke any Tcl procedures on the interpreter, since its state is partially deleted. All that trace procedures should do under these circumstances is to clean up and free their own internal data structures.

BUGS
Tcl does not do any error checking to prevent trace procedures from misusing the interpreter during traces with TCL_INTERP_DESTROYED set.

Keywords: clientData, trace, command

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2002 Donal K. Fellows
NAME

Tcl_Concat - concatenate a collection of strings

SYNOPSIS

#include <tcl.h>
const char *
Tcl_Concat(argc, argv)

ARGUMENTS

int argc (in) Number of strings.

const char *const argv[] (in) Array of strings to concatenate. Must have argc entries.

DESCRIPTION

Tcl_Concat is a utility procedure used by several of the Tcl commands. Given a collection of strings, it concatenates them together into a single string, with the original strings separated by spaces. This procedure behaves differently than Tcl_Merge, in that the arguments are simply concatenated: no effort is made to ensure proper list structure. However, in most common usage the arguments will all be proper lists themselves; if this is true, then the result will also have proper list structure.

Tcl_Concat eliminates leading and trailing white space as it copies strings from argv to the result. If an element of argv consists of nothing
but white space, then that string is ignored entirely. This white-space removal was added to make the output of the `concat` command cleaner-looking.

The result string is dynamically allocated using `Tcl_Alloc`; the caller must eventually release the space by calling `Tcl_Free`.

**SEE ALSO**

`Tcl_ConcatObj`

**KEYWORDS**

`concatenate`, `strings`

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tcl_NewDoubleObj, Tcl_SetDoubleObj, Tcl_GetDoubleFromObj - manipulate Tcl objects as floating-point values

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_NewDoubleObj(doubleValue)
Tcl_SetDoubleObj(objPtr, doubleValue)
int
Tcl_GetDoubleFromObj(interp, objPtr, doublePtr)

ARGUMENTS

DESCRIPTION

SEE ALSO

KEYWORDS
## DESCRIPTION

These procedures are used to create, modify, and read Tcl objects that hold double-precision floating-point values.

**Tcl NewDoubleObj** creates and returns a new Tcl object initialized to the double value `doubleValue`. The returned Tcl object is unshared.

**Tcl SetDoubleObj** sets the value of an existing Tcl object pointed to by `objPtr` to the double value `doubleValue`. The `objPtr` argument must point to an unshared Tcl object. Any attempt to set the value of a shared Tcl object violates Tcl's copy-on-write policy. Any existing string representation or internal representation in the unshared Tcl object will be freed as a consequence of setting the new value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>double <code>doubleValue</code> (in)</td>
<td>A double-precision floating-point value used to initialize or set a Tcl object.</td>
</tr>
<tr>
<td>Tcl_Obj *<code>objPtr</code> (in/out)</td>
<td>For <strong>Tcl SetDoubleObj</strong>, this points to the object in which to store a double value. For <strong>Tcl GetDoubleFromObj</strong>, this refers to the object from which to retrieve a double value.</td>
</tr>
<tr>
<td>Tcl_Interp *<code>interp</code> (in/out)</td>
<td>When non-NULL, an error message is left here when double value retrieval fails.</td>
</tr>
<tr>
<td>double *<code>doublePtr</code> (out)</td>
<td>Points to place to store the double value obtained from <code>objPtr</code>.</td>
</tr>
</tbody>
</table>
Tcl_GetDoubleFromObj attempts to retrieve a double value from the Tcl object objPtr. If the attempt succeeds, then TCL_OK is returned, and the double value is written to the storage pointed to by doublePtr. If the attempt fails, then TCL_ERROR is returned, and if interp is non-NULL, an error message is left in interp. The Tcl_ObjType of objPtr may be changed to make subsequent calls to Tcl_GetDoubleFromObj more efficient.

SEE ALSO

Tcl_NewObj, Tcl_DecrRefCount, Tcl_IncrRefCount, Tcl_GetObjResult

KEYWORDS

double, double object, double type, internal representation, object, object type, string representation

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996-1997 Sun Microsystems, Inc.
NAME
Tcl_ConditionNotify, Tcl_ConditionWait, Tcl_ConditionFinalize,
Tcl_GetThreadData, Tcl_MutexLock, Tcl_MutexUnlock,
Tcl_MutexFinalize, Tcl_CreateThread, Tcl_JoinThread - Tcl thread support

SYNOPSIS
#include <tcl.h>

void
Tcl_ConditionNotify(condPtr)

void
Tcl_ConditionWait(condPtr, mutexPtr, timePtr)

void
Tcl_ConditionFinalize(condPtr)

Void *
Tcl_GetThreadData(keyPtr, size)

void
Tcl_MutexLock(mutexPtr)

void
Tcl_MutexUnlock(mutexPtr)

void
Tcl_MutexFinalize(mutexPtr)

int
Tcl_CreateThread(idPtr, threadProc, clientData, stackSize, flags)

int
Tcl_JoinThread(id, result)

ARGUMENTS
INTRODUCTION
DESCRIPTION
SYNCHRONIZATION AND COMMUNICATION
INITIALIZE
SCRIPT-LEVEL ACCESS TO THREADS
SEE ALSO
KEYWORDS

NAME

Tcl_ConditionNotify, Tcl_ConditionWait, Tcl_ConditionFinalize, Tcl_GetThreadData, Tcl_MutexLock, Tcl_MutexUnlock, Tcl_MutexFinalize, Tcl_CreateThread, Tcl_JoinThread - Tcl thread support

SYNOPSIS

#include <tcl.h>
void
Tcl_ConditionNotify(condPtr)
void
Tcl_ConditionWait(condPtr, mutexPtr, timePtr)
void
Tcl_ConditionFinalize(condPtr)
Void *
Tcl_GetThreadData(keyPtr, size)
void
Tcl_MutexLock(mutexPtr)
void
Tcl_MutexUnlock(mutexPtr)
void
Tcl_MutexFinalize(mutexPtr)
int
Tcl_CreateThread(idPtr, threadProc, clientData, stackSize, flags)
int
Tcl_JoinThread(id, result)

ARGUMENTS

Tcl_Condition *condPtr (in) A condition variable, which must be associated with a mutex lock.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Mutex *mutexPtr (in)</code></td>
<td>A mutex lock.</td>
</tr>
<tr>
<td><code>Tcl_Time *timePtr (in)</code></td>
<td>A time limit on the condition wait. NULL to wait forever. Note that a polling value of 0 seconds does not make much sense.</td>
</tr>
<tr>
<td><code>Tcl_ThreadDataKey *keyPtr (in)</code></td>
<td>This identifies a block of thread local storage. The key should be static and process-wide, yet each thread will end up associating a different block of storage with this key.</td>
</tr>
<tr>
<td><code>int *size (in)</code></td>
<td>The size of the thread local storage block. This amount of data is allocated and initialized to zero the first time each thread calls <code>.Tcl_GetThreadData</code>.</td>
</tr>
<tr>
<td><code>Tcl_ThreadId *idPtr (out)</code></td>
<td>The referred storage will contain the id of the newly created thread as returned by the operating system.</td>
</tr>
<tr>
<td><code>Tcl_ThreadId id (in)</code></td>
<td>Id of the thread waited upon.</td>
</tr>
<tr>
<td><code>Tcl_ThreadCreateProc threadProc (in)</code></td>
<td>This procedure will act as the <code>main()</code> of the newly created thread. The specified <code>clientData</code> will be</td>
</tr>
</tbody>
</table>
### INTRODUCTION

Beginning with the 8.1 release, the Tcl core is thread safe, which allows you to incorporate Tcl into multithreaded applications without customizing the Tcl core. To enable Tcl multithreading support, you must include the `--enable-threads` option to `configure` when you configure and compile your Tcl core.

An important constraint of the Tcl threads implementation is that *only the thread that created a Tcl interpreter can use that interpreter*. In other words, multiple threads can not access the same Tcl interpreter. (However, a single thread can safely create and use multiple interpreters.)

### DESCRIPTION

Tcl provides `Tcl_CreateThread` for creating threads. The caller can determine the size of the stack given to the new thread and modify the

<table>
<thead>
<tr>
<th>ClientData clientData (in)</th>
<th>Arbitrary information. Passed as sole argument to the <code>threadProc</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>int stackSize (in)</td>
<td>The size of the stack given to the new thread.</td>
</tr>
<tr>
<td>int flags (in)</td>
<td>Bitmask containing flags allowing the caller to modify behaviour of the new thread.</td>
</tr>
<tr>
<td>int *result (out)</td>
<td>The referred storage is used to place the exit code of the thread waited upon into it.</td>
</tr>
</tbody>
</table>
behaviour through the supplied flags. The value **TCL_THREAD_STACK_DEFAULT** for the stackSize indicates that the default size as specified by the operating system is to be used for the new thread. As for the flags, currently only the values **TCL_THREAD_NOFLAGS** and **TCL_THREAD_JOINABLE** are defined. The first of them invokes the default behaviour with no specialties. Using the second value marks the new thread as *joinable*. This means that another thread can wait for the such marked thread to exit and join it.

Restrictions: On some UNIX systems the pthread-library does not contain the functionality to specify the stack size of a thread. The specified value for the stack size is ignored on these systems. Windows currently does not support joinable threads. This flag value is therefore ignored on this platform.

Tcl provides the **Tcl_ExitThread** and **Tcl_FinalizeThread** functions for terminating threads and invoking optional per-thread exit handlers. See the **Tcl.Exit** page for more information on these procedures.

The **Tcl_JoinThread** function is provided to allow threads to wait upon the exit of another thread, which must have been marked as joinable through usage of the **TCL_THREAD_JOINABLE**-flag during its creation via **Tcl_CreateThread**.

Trying to wait for the exit of a non-joinable thread or a thread which is already waited upon will result in an error. Waiting for a joinable thread which already exited is possible, the system will retain the necessary information until after the call to **Tcl_JoinThread**. This means that not calling **Tcl_JoinThread** for a joinable thread will cause a memory leak.

The **Tcl_GetThreadData** call returns a pointer to a block of thread-private data. Its argument is a key that is shared by all threads and a size for the block of storage. The storage is automatically allocated and initialized to all zeros the first time each thread asks for it. The storage is automatically deallocated by **Tcl_FinalizeThread**.

**SYNCHRONIZATION AND COMMUNICATION**
Tcl provides `Tcl_ThreadQueueEvent` and `Tcl_ThreadAlert` for handling event queuing in multithreaded applications. See the `Notifier` manual page for more information on these procedures.

A mutex is a lock that is used to serialize all threads through a piece of code by calling `Tcl_MutexLock` and `Tcl_MutexUnlock`. If one thread holds a mutex, any other thread calling `Tcl_MutexLock` will block until `Tcl_MutexUnlock` is called. A mutex can be destroyed after its use by calling `Tcl_MutexFinalize`. The result of locking a mutex twice from the same thread is undefined. On some platforms it will result in a deadlock. The `Tcl_MutexLock`, `Tcl_MutexUnlock` and `Tcl_MutexFinalize` procedures are defined as empty macros if not compiling with threads enabled. For declaration of mutexes the `TCL_DECLARE_MUTEX` macro should be used. This macro assures correct mutex handling even when the core is compiled without threads enabled.

A condition variable is used as a signaling mechanism: a thread can lock a mutex and then wait on a condition variable with `Tcl.ConditionWait`. This atomically releases the mutex lock and blocks the waiting thread until another thread calls `Tcl.ConditionNotify`. The caller of `Tcl.ConditionNotify` should have the associated mutex held by previously calling `Tcl_MutexLock`, but this is not enforced. Notifying the condition variable unblocks all threads waiting on the condition variable, but they do not proceed until the mutex is released with `Tcl_MutexUnlock`. The implementation of `Tcl.ConditionWait` automatically locks the mutex before returning.

The caller of `Tcl.ConditionWait` should be prepared for spurious notifications by calling `Tcl.ConditionWait` within a while loop that tests some invariant.

A condition variable can be destroyed after its use by calling `Tcl.ConditionFinalize`.

The `Tcl.ConditionNotify`, `Tcl.ConditionWait` and `Tcl.ConditionFinalize` procedures are defined as empty macros if not compiling with threads enabled.
INITIALIZATION

All of these synchronization objects are self-initializing. They are implemented as opaque pointers that should be NULL upon first use. The mutexes and condition variables are either cleaned up by process exit handlers (if living that long) or explicitly by calls to `Tcl_MutexFinalize` or `Tcl_ConditionFinalize`. Thread local storage is reclaimed during `Tcl_FinalizeThread`.

SCRIPT-LEVEL ACCESS TO THREADS

Tcl provides no built-in commands for scripts to use to create, manage, or join threads, nor any script-level access to mutex or condition variables. It provides such facilities only via C interfaces, and leaves it up to packages to expose these matters to the script level. One such package is the Thread package.

SEE ALSO

`Tcl_GetCurrentThread`, `Tcl_ThreadQueueEvent`, `Tcl_ThreadAlert`, `Tcl_ExitThread`, `Tcl_FinalizeThread`, `Tcl_CreateThreadExitHandler`, `Tcl_DeleteThreadExitHandler`, Thread

KEYWORDS

thread, mutex, condition variable, thread local storage

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1999 Scriptics Corporation
Copyright © 1998 Sun Microsystems, Inc.
NAME
Tcl_CreateEnsemble, Tcl_FindEnsemble,
Tcl_GetEnsembleFlags, Tcl_GetEnsembleMappingDict,
Tcl_GetEnsembleNamespace,
Tcl_GetEnsembleUnknownHandler,
Tcl_GetEnsembleSubcommandList, Tcl_IsEnsemble,
Tcl_SetEnsembleFlags, Tcl_SetEnsembleMappingDict,
Tcl_SetEnsembleSubcommandList,
Tcl_SetEnsembleUnknownHandler - manipulate ensemble commands

SYNOPSIS
#include <tcl.h>
Tcl_Command
Tcl_CreateEnsemble(interp, name, namespacePtr, ensFlags)
Tcl_Command
Tcl_FindEnsemble(interp, cmdNameObj, flags)
int
Tcl_IsEnsemble(token)
int
Tcl_GetEnsembleFlags(interp, token, ensFlagsPtr)
int
Tcl_SetEnsembleFlags(interp, token, ensFlags)
int
Tcl_GetEnsembleMappingDict(interp, token, dictObjPtr)
int
Tcl_SetEnsembleMappingDict(interp, token, dictObj)
int
Tcl_GetEnsembleSubcommandList(interp, token, listObjPtr)
int
Tcl_SetEnsembleSubcommandList(interp, token, listObj)
int
Tcl_GetEnsembleUnknownHandler(interp, token, listObjPtr)
int
Tcl_SetEnsembleUnknownHandler(interp, token, listObj)
int
Tcl_GetEnsembleNamespace(interp, token, namespacePtrPtr)

ARGUMENTS
DESCRIPTION
ENSEMBLE PROPERTIES
flags (read-write)
mapping dictionary (read-write)
subcommand list (read-write)
unknown subcommand handler command prefix (read-write)
bound namespace (read-only)

SEE ALSO

NAME

Tcl_CreateEnsemble, Tcl_FindEnsemble, Tcl_GetEnsembleFlags,
Tcl_GetEnsembleMappingDict, Tcl_GetEnsembleNamespace,
Tcl_GetEnsembleUnknownHandler, Tcl_GetEnsembleSubcommandList,
Tcl_IsEnsemble, Tcl_SetEnsembleFlags, Tcl_SetEnsembleMappingDict,
Tcl_SetEnsembleSubcommandList, Tcl_SetEnsembleUnknownHandler
- manipulate ensemble commands

SYNOPSIS

#include <tcl.h>
Tcl_Command
Tcl_CreateEnsemble(interp, name, namespacePtr, ensFlags)
Tcl_Command
Tcl_FindEnsemble(interp, cmdNameObj, flags)
int
Tcl_IsEnsemble(token)
int
Tcl_GetEnsembleFlags(interp, token, ensFlagsPtr)
int
Tcl_SetEnsembleFlags(\textit{interp}, \textit{token}, \textit{ensFlags})
\textbf{int}

Tcl_GetEnsembleMappingDict(\textit{interp}, \textit{token}, \textit{dictObjPtr})
\textbf{int}

Tcl_SetEnsembleMappingDict(\textit{interp}, \textit{token}, \textit{dictObj})
\textbf{int}

Tcl_GetEnsembleSubcommandList(\textit{interp}, \textit{token}, \textit{listObjPtr})
\textbf{int}

Tcl_SetEnsembleSubcommandList(\textit{interp}, \textit{token}, \textit{listObj})
\textbf{int}

Tcl_GetEnsembleUnknownHandler(\textit{interp}, \textit{token}, \textit{listObjPtr})
\textbf{int}

Tcl_SetEnsembleUnknownHandler(\textit{interp}, \textit{token}, \textit{listObj})
\textbf{int}

Tcl_GetEnsembleNamespace(\textit{interp}, \textit{token}, \textit{namespacePtrPtr})

\textbf{ARGUMENTS}

\begin{tabular}{|l|l|}
\hline
\textbf{Tcl\_Interp} *\textit{interp} (in/out) & The interpreter in which the ensemble is to be created or found. Also where error result messages are written. The functions whose names start with \textbf{Tcl\_GetEnsemble} may have a NULL for the \textit{interp}, but all other functions must not. \\
\hline
\textbf{const char} *\textit{name} (in) & The name of the ensemble command to be created. \\
\hline
\textbf{Tcl\_Namespace} *\textit{namespacePtr} (in) & The namespace to which the ensemble command is to be bound, or NULL for the current namespace. \\
\hline
\end{tabular}
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <code>ensFlags</code> (in)</td>
<td>An ORed set of flag bits describing the basic configuration of the ensemble. Currently only one bit has meaning, <code>TCL_ENSEMBLE_PREFIX</code>, which is present when the ensemble command should also match unambiguous prefixes of subcommands.</td>
</tr>
<tr>
<td>Tcl_Obj <code>cmdNameObj</code> (in)</td>
<td>A value holding the name of the ensemble command to look up.</td>
</tr>
<tr>
<td>int <code>flags</code> (in)</td>
<td>An ORed set of flag bits controlling the behavior of <code>Tcl_FindEnsemble</code>. Currently only <code>TCL_LEAVE_ERR_MSG</code> is supported.</td>
</tr>
<tr>
<td>Tcl_Command <code>token</code> (in)</td>
<td>A normal command token that refers to an ensemble command, or which you wish to use for testing as an ensemble command in <code>Tcl_IsEnsemble</code>.</td>
</tr>
<tr>
<td>int <code>*ensFlagsPtr</code> (out)</td>
<td>Pointer to a variable into which to write the current ensemble flag bits; currently only the bit <code>TCL_ENSEMBLE_PREFIX</code> is defined.</td>
</tr>
<tr>
<td>**Tcl_Obj <strong>dictObj (in)</strong></td>
<td>A dictionary value to use for the subcommand to implementation command prefix mapping dictionary in the ensemble. May be NULL if the mapping dictionary is to be removed.</td>
</tr>
<tr>
<td>**Tcl_Obj <strong>dictObjPtr (out)</strong></td>
<td>Pointer to a variable into which to write the current ensemble mapping dictionary.</td>
</tr>
<tr>
<td>**Tcl_Obj <strong>listObj (in)</strong></td>
<td>A list value to use for the defined list of subcommands in the dictionary or the unknown subcommand handler command prefix. May be NULL if the subcommand list or unknown handler are to be removed.</td>
</tr>
<tr>
<td>**Tcl_Obj <strong>listObjPtr (out)</strong></td>
<td>Pointer to a variable into which to write the current defined list of subcommands or the current unknown handler prefix.</td>
</tr>
<tr>
<td>**Tcl_Namespace <strong>namespacePtrPtr (out)</strong></td>
<td>Pointer to a variable into which to write the handle of the namespace to which the ensemble is bound.</td>
</tr>
</tbody>
</table>
An ensemble is a command, bound to some namespace, which consists of a collection of subcommands implemented by other Tcl commands. The first argument to the ensemble command is always interpreted as a selector that states what subcommand to execute.

Ensembles are created using `Tcl_CreateEnsemble`, which takes four arguments: the interpreter to work within, the name of the ensemble to create, the namespace within the interpreter to bind the ensemble to, and the default set of ensemble flags. The result of the function is the command token for the ensemble, which may be used to further configure the ensemble using the API described below in **ENSEMBLE PROPERTIES**.

Given the name of an ensemble command, the token for that command may be retrieved using `Tcl_FindEnsemble`. If the given command name (in `cmdNameObj`) does not refer to an ensemble command, the result of the function is NULL and (if the TCL_LEAVE_ERR_MSG bit is set in `flags`) an error message is left in the interpreter result.

A command token may be checked to see if it refers to an ensemble using `Tcl_IsEnsemble`. This returns 1 if the token refers to an ensemble, or 0 otherwise.

**ENSEMBLE PROPERTIES**

Every ensemble has four read-write properties and a read-only property. The properties are:

- **flags** (read-write)
  The set of flags for the ensemble, expressed as a bit-field. Currently, the only public flag is TCL_ENSEMBLE_PREFIX which is set when unambiguous prefixes of subcommands are permitted to be resolved to implementations as well as exact matches. The flags may be read and written using `Tcl_GetEnsembleFlags` and `Tcl_SetEnsembleFlags` respectively. The result of both of those functions is a Tcl result code (TCL_OK, or TCL_ERROR if the token
mapping dictionary (read-write)
A dictionary containing a mapping from subcommand names to lists of words to use as a command prefix (replacing the first two words of the command which are the ensemble command itself and the subcommand name), or NULL if every subcommand is to be mapped to the command with the same unqualified name in the ensemble's bound namespace. Defaults to NULL. May be read and written using Tcl_GetEnsembleMappingDict and Tcl_SetEnsembleMappingDict respectively. The result of both of those functions is a Tcl result code (TCL_OK, or TCL_ERROR if the token does not refer to an ensemble) and the dictionary obtained from Tcl_GetEnsembleMappingDict should always be treated as immutable even if it is unshared.

subcommand list (read-write)
A list of all the subcommand names for the ensemble, or NULL if this is to be derived from either the keys of the mapping dictionary (see above) or (if that is also NULL) from the set of commands exported by the bound namespace. May be read and written using Tcl_GetEnsembleSubcommandList and Tcl_SetEnsembleSubcommandList respectively. The result of both of those functions is a Tcl result code (TCL_OK, or TCL_ERROR if the token does not refer to an ensemble) and the list obtained from Tcl_GetEnsembleSubcommandList should always be treated as immutable even if it is unshared.

unknown subcommand handler command prefix (read-write)
A list of words to prepend on the front of any subcommand when the subcommand is unknown to the ensemble (according to the current prefix handling rule); see the namespace ensemble command for more details. If NULL, the default behavior - generate a suitable error message - will be used when an unknown subcommand is encountered. May be read and written using Tcl_GetEnsembleUnknownHandler and Tcl_SetEnsembleUnknownHandler respectively. The result of both functions is a Tcl result code (TCL_OK, or TCL_ERROR if the
token does not refer to an ensemble) and the list obtained from `Tcl_GetEnsembleUnknownHandler` should always be treated as immutable even if it is unshared.

**bound namespace** (read-only)
The namespace to which the ensemble is bound; when the namespace is deleted, so too will the ensemble, and this namespace is also the namespace whose list of exported commands is used if both the mapping dictionary and the subcommand list properties are NULL. May be read using `Tcl_GetEnsembleNamespace` which returns a Tcl result code (TCL_OK, or TCL_ERROR if the token does not refer to an ensemble).

**SEE ALSO**

namespace, `Tcl_DeleteCommandFromToken`
NAME
Tcl_CreateChannelHandler, Tcl_DeleteChannelHandler - call a procedure when a channel becomes readable or writable

SYNOPSIS
#include <tcl.h>
void
Tcl_CreateChannelHandler(channel, mask, proc, clientData)
void
Tcl_DeleteChannelHandler(channel, proc, clientData)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS
**DESCRIPTION**

**Tcl_CreateChannelHandler** arranges for *proc* to be called in the future whenever input or output becomes possible on the channel identified by *channel*, or whenever an exceptional condition exists for *channel*. The conditions of interest under which *proc* will be invoked are specified by the *mask* argument. See the manual entry for **fileevent** for a precise description of what it means for a channel to be readable or writable. *Proc* must conform to the following prototype:

```c
typedef void Tcl_ChannelProc(
    ClientData clientData, 
    int mask);
```

The *clientData* argument is the same as the value passed to **Tcl_CreateChannelHandler** when the handler was created. Typically,
*clientData* points to a data structure containing application-specific information about the channel. *Mask* is an integer mask indicating which of the requested conditions actually exists for the channel; it will contain a subset of the bits from the *mask* argument to *Tcl_CreateChannelHandler* when the handler was created.

Each channel handler is identified by a unique combination of *channel*, *proc* and *clientData*. There may be many handlers for a given channel as long as they do not have the same *channel*, *proc*, and *clientData*. If *Tcl_CreateChannelHandler* is invoked when there is already a handler for *channel*, *proc*, and *clientData*, then no new handler is created; instead, the *mask* is changed for the existing handler.

*Tcl_DeleteChannelHandler* deletes a channel handler identified by *channel*, *proc* and *clientData*; if no such handler exists, the call has no effect.

Channel handlers are invoked via the Tcl event mechanism, so they are only useful in applications that are event-driven. Note also that the conditions specified in the *mask* argument to *proc* may no longer exist when *proc* is invoked: for example, if there are two handlers for *TCL_READABLE* on the same channel, the first handler could consume all of the available input so that the channel is no longer readable when the second handler is invoked. For this reason it may be useful to use nonblocking I/O on channels for which there are event handlers.

**SEE ALSO**

Notifier, *Tcl_CreateChannel*, *Tcl_OpenFileChannel*, *vwait(n)*.

**KEYWORDS**

blocking, callback, channel, events, handler, nonblocking.

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
NAME
Tcl_CreateCloseHandler, Tcl_DeleteCloseHandler - arrange for callbacks when channels are closed

SYNOPSIS
#include <tcl.h>
void
Tcl_CreateCloseHandler(channel, proc, clientData)
void
Tcl_DeleteCloseHandler(channel, proc, clientData)

ARGUMENTS

DESCRIPTION

SEE ALSO

KEYWORDS

NAME
Tcl_CreateCloseHandler, Tcl_DeleteCloseHandler - arrange for callbacks when channels are closed

SYNOPSIS
#include <tcl.h>
void
Tcl_CreateCloseHandler(channel, proc, clientData)
void
Tcl_DeleteCloseHandler(channel, proc, clientData)

ARGUMENTS

Tcl_Channel channel (in) The channel for which to create or delete a close callback.
**DESCRIPTION**

**Tcl_CreateCloseHandler** arranges for `proc` to be called when `channel` is closed with **Tcl_Close** or **Tcl_UnregisterChannel**, or using the Tcl `close` command. *Proc* should match the following prototype:

```c
typedef void Tcl_CloseProc(
    ClientData clientData);
```

The `clientData` is the same as the value provided in the call to **Tcl_CreateCloseHandler**.

**Tcl_DeleteCloseHandler** removes a close callback for `channel`. The `proc` and `clientData` identify which close callback to remove; **Tcl_DeleteCloseHandler** does nothing if its `proc` and `clientData` arguments do not match the `proc` and `clientData` for a close handler for `channel`.

**SEE ALSO**

close, Tcl_Close, Tcl_UnregisterChannel

**KEYWORDS**

callback, channel closing

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tcl_SetErrno, Tcl_GetErrno, Tcl_Errnold, Tcl_ErrnoMsg - manipulate errno to store and retrieve error codes

SYNOPSIS
#include <tcl.h>

void
Tcl_SetErrno(errorCode)
int
Tcl_GetErrno()
const char *
Tcl_Errnold()
const char *
Tcl_ErrnoMsg(errorCode)

ARGUMENTS

DESCRIPTION

KEYWORDS

NAME
Tcl_SetErrno, Tcl_GetErrno, Tcl_Errnold, Tcl_ErrnoMsg - manipulate errno to store and retrieve error codes

SYNOPSIS
#include <tcl.h>

void
Tcl_SetErrno(errorCode)
int
Tcl_GetErrno()
const char *
Tcl_Errnold()
const char *
Tcl_ErrnoMsg(errorCode)
**Tcl_ErrnoMsg(errorCode)**

**ARGUMENTS**

| int errorCode (in) | A POSIX error code such as ENOENT. |

**DESCRIPTION**

**Tcl_SetErrno** and **Tcl_GetErrno** provide portable access to the *errno* variable, which is used to record a POSIX error code after system calls and other operations such as **Tcl_Gets**. These procedures are necessary because global variable accesses cannot be made across module boundaries on some platforms.

**Tcl_SetErrno** sets the *errno* variable to the value of the *errorCode* argument C procedures that wish to return error information to their callers via *errno* should call **Tcl_SetErrno** rather than setting *errno* directly.

**Tcl_GetErrno** returns the current value of *errno*. Procedures wishing to access *errno* should call this procedure instead of accessing *errno* directly.

**Tcl_ErrnoId** and **Tcl_ErrnoMsg** return string representations of *errno* values. **Tcl_ErrnoId** returns a machine-readable textual identifier such as “EACCES” that corresponds to the current value of *errno*. **Tcl_ErrnoMsg** returns a human-readable string such as “permission denied” that corresponds to the value of its *errorCode* argument. The *errorCode* argument is typically the value returned by **Tcl_GetErrno**. The strings returned by these functions are statically allocated and the caller must not free or modify them.

**KEYWORDS**

errno, error_code, global variables
NAME

Tcl_CreateCommand - implement new commands in C

SYNOPSIS

```c
#include <tcl.h>
Tcl_Command
Tcl_CreateCommand(interp, cmdName, proc, clientData, deleteProc)
```

ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_Interp *interp (in)</td>
<td>Interpreter in which to create new command.</td>
</tr>
<tr>
<td>const char *cmdName (in)</td>
<td>Name of command.</td>
</tr>
<tr>
<td>Tcl_CmdProc *proc (in)</td>
<td>Implementation of new command: <code>proc</code> will be called whenever <code>cmdName</code> is invoked as a command.</td>
</tr>
<tr>
<td>ClientData clientData (in)</td>
<td>Arbitrary one-word value to pass to <code>proc</code> and <code>deleteProc</code>.</td>
</tr>
<tr>
<td>Tcl_CmdDeleteProc *deleteProc (in)</td>
<td>Procedure to call before <code>cmdName</code> is deleted from the interpreter; allows for command-specific cleanup. If NULL, then no</td>
</tr>
</tbody>
</table>
DESCRIPTION

**Tcl_CreateCommand** defines a new command in *interp* and associates it with procedure *proc* such that whenever *cmdName* is invoked as a Tcl command (via a call to **Tcl_Eval**) the Tcl interpreter will call *proc* to process the command. It differs from **Tcl_CreateObjCommand** in that a new string-based command is defined; that is, a command procedure is defined that takes an array of argument strings instead of objects. The object-based command procedures registered by **Tcl_CreateObjCommand** can execute significantly faster than the string-based command procedures defined by **Tcl_CreateCommand**. This is because they take Tcl objects as arguments and those objects can retain an internal representation that can be manipulated more efficiently. Also, Tcl's interpreter now uses objects internally. In order to invoke a string-based command procedure registered by **Tcl_CreateCommand**, it must generate and fetch a string representation from each argument object before the call and create a new Tcl object to hold the string result returned by the string-based command procedure. New commands should be defined using **Tcl_CreateObjCommand**. We support **Tcl_CreateCommand** for backwards compatibility.

The procedures **Tcl_DeleteCommand**, **Tcl_GetCommandInfo**, and **Tcl_SetCommandInfo** are used in conjunction with **Tcl_CreateCommand**.

**Tcl_CreateCommand** will delete an existing command *cmdName*, if one is already associated with the interpreter. It returns a token that may be used to refer to the command in subsequent calls to **Tcl_GetCommandName**. If *cmdName* contains any :: namespace qualifiers, then the command is added to the specified namespace; otherwise the command is added to the global namespace. If **Tcl_CreateCommand** is called for an interpreter that is in the process of being deleted, then it does not create a new command and it returns
Proc should have arguments and result that match the type Tcl_CmdProc:

```c
typedef int Tcl_CmdProc(
    ClientData clientData,
    Tcl_Interp *interp,
    int argc,
    const char *argv[]);
```

When proc is invoked the clientData and interp parameters will be copies of the clientData and interp arguments given to Tcl_CreateCommand. Typically, clientData points to an application-specific data structure that describes what to do when the command procedure is invoked. Argc and argv describe the arguments to the command, argc giving the number of arguments (including the command name) and argv giving the values of the arguments as strings. The argv array will contain argc+1 values; the first argc values point to the argument strings, and the last value is NULL. Note that the argument strings should not be modified as they may point to constant strings or may be shared with other parts of the interpreter.

Note that the argument strings are encoded in normalized UTF-8 since version 8.1 of Tcl.

Proc must return an integer code that is expected to be one of TCL_OK, TCL_ERROR, TCL_RETURN, TCL_BREAK, or TCL_CONTINUE. See the Tcl overview man page for details on what these codes mean. Most normal commands will only return TCL_OK or TCL_ERROR. In addition, proc must set the interpreter result to point to a string value; in the case of a TCL_OK return code this gives the result of the command, and in the case of TCL_ERROR it gives an error message. The Tcl_SetResult procedure provides an easy interface for setting the return value; for complete details on how the interpreter result field is managed, see the Tcl_Interp man page. Before invoking a command procedure, Tcl_Eval sets the interpreter result to point to an empty string, so simple commands can return an empty result by doing
nothing at all.

The contents of the \textit{argv} array belong to Tcl and are not guaranteed to persist once \textit{proc} returns: \textit{proc} should not modify them, nor should it set the interpreter result to point anywhere within the \textit{argv} values. Call \texttt{Tcl\_SetResult} with status \texttt{TCL\_VOLATILE} if you want to return something from the \textit{argv} array.

\textit{DeleteProc} will be invoked when (if) \textit{cmdName} is deleted. This can occur through a call to \texttt{Tcl\_DeleteCommand} or \texttt{Tcl\_DeleteInterp}, or by replacing \textit{cmdName} in another call to \texttt{Tcl\_CreateCommand}. \textit{DeleteProc} is invoked before the command is deleted, and gives the application an opportunity to release any structures associated with the command. \textit{DeleteProc} should have arguments and result that match the type \texttt{Tcl\_CmdDeleteProc}:

\begin{verbatim}
typedef void Tcl_CmdDeleteProc(
    ClientData clientData);
\end{verbatim}

The \textit{clientData} argument will be the same as the \textit{clientData} argument passed to \texttt{Tcl\_CreateCommand}.

\textbf{SEE ALSO}

\texttt{Tcl\_CreateObjCommand, Tcl\_DeleteCommand, Tcl\_GetCommandInfo, Tcl\_SetCommandInfo, Tcl\_GetCommandName, Tcl\_SetObjResult}

\textbf{KEYWORDS}

\texttt{bind, command, create, delete, interpreter, namespace}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME
Tcl_InitHashTable, Tcl_InitCustomHashTable,
Tcl_InitObjHashTable, Tcl_DeleteHashTable,
Tcl_CreateHashEntry, Tcl_DeleteHashEntry,
Tcl_FindHashEntry, Tcl_GetHashValue, Tcl_SetHashValue,
Tcl_GetHashKey, Tcl_FirstHashEntry, Tcl_NextHashEntry,
Tcl_HashStats - procedures to manage hash tables

SYNOPSIS
#include <tcl.h>
Tcl_InitHashTable(tablePtr, keyType)
Tcl_InitCustomHashTable(tablePtr, keyType, typePtr)
Tcl_InitObjHashTable(tablePtr)
Tcl_DeleteHashTable(tablePtr)
Tcl_HashEntry *
Tcl_CreateHashEntry(tablePtr, key, newPtr)
Tcl_DeleteHashEntry(entryPtr)
Tcl_HashEntry *
Tcl_FindHashEntry(tablePtr, key)
ClientData
Tcl_GetHashValue(entryPtr)
Tcl_SetHashValue(entryPtr, value)
char *
Tcl_GetHashKey(tablePtr, entryPtr)
Tcl_HashEntry *
Tcl_FirstHashEntry(tablePtr, searchPtr)
Tcl_HashEntry *
Tcl_NextHashEntry(searchPtr)
char *
Tcl_HashStats(tablePtr)

ARGUMENTS
DESCRIPTION
THE TCL_HASHKEYTYPE STRUCTURE

TCL_HASH_KEY_RANDOMIZE_HASH
TCL_HASH_KEY_SYSTEM_HASH

KEYWORDS

NAME

Tcl_InitHashTable, Tcl_InitCustomHashTable, Tcl_InitObjHashTable,
Tcl_DeleteHashTable, Tcl_CreateHashEntry, Tcl_DeleteHashEntry,
Tcl_FindHashEntry, Tcl_GetHashValue, Tcl_SetHashValue,
Tcl_GetHashKey, Tcl_FirstHashEntry, Tcl_NextHashEntry,
Tcl_HashStats - procedures to manage hash tables

SYNOPSIS

#include <tcl.h>
Tcl_InitHashTable( tablePtr, keyType)
Tcl_InitCustomHashTable( tablePtr, keyType, typePtr)
Tcl_InitObjHashTable( tablePtr)
Tcl_DeleteHashTable( tablePtr)
Tcl_HashEntry *
Tcl_CreateHashEntry( tablePtr, key, newPtr)
Tcl_DeleteHashEntry( entryPtr)
Tcl_HashEntry *
Tcl_FindHashEntry( tablePtr, key)
ClientData
Tcl_GetHashValue( entryPtr)
Tcl_SetHashValue( entryPtr, value)
char *
Tcl_GetHashKey( tablePtr, entryPtr)
Tcl_HashEntry *
Tcl_FirstHashEntry( tablePtr, searchPtr)
Tcl_HashEntry *
**ARGUMENTS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_HashTable *tablePtr (in)</code></td>
<td>Address of hash table structure (for all procedures but <code>Tcl_InitHashTable</code>, this must have been initialized by previous call to <code>Tcl_InitHashTable</code>).</td>
</tr>
<tr>
<td><code>int keyType (in)</code></td>
<td>Kind of keys to use for new hash table. Must be either <code>TCL_STRING_KEYS</code>, <code>TCL_ONE_WORD_KEYS</code>, <code>TCL_CUSTOM_TYPE_KEYS</code>, <code>TCL_CUSTOM_PTR_KEYS</code>, or an integer value greater than 1.</td>
</tr>
<tr>
<td><code>Tcl_HashKeyType *typePtr (in)</code></td>
<td>Address of structure which defines the behaviour of the hash table.</td>
</tr>
<tr>
<td><code>const char *key (in)</code></td>
<td>Key to use for probe into table. Exact form depends on <code>keyType</code> used to create table.</td>
</tr>
<tr>
<td><code>int *newPtr (out)</code></td>
<td>The word at <code>*newPtr</code> is set to 1 if a new entry was created and 0 if there was already an entry for <code>key</code>.</td>
</tr>
</tbody>
</table>
### DESCRIPTION

A hash table consists of zero or more entries, each consisting of a key and a value. Given the key for an entry, the hashing routines can very quickly locate the entry, and hence its value. There may be at most one entry in a hash table with a particular key, but many entries may have the same value. Keys can take one of four forms: strings, one-word values, integer arrays, or custom keys defined by a Tcl_HashKeyType structure (See section **THE TCL_HASHKEYTYPE STRUCTURE** below). All of the keys in a given table have the same form, which is specified when the table is initialized.

The value of a hash table entry can be anything that fits in the same space as a “char *” pointer. Values for hash table entries are managed entirely by clients, not by the hash module itself. Typically each entry's value is a pointer to a data structure managed by client code.

Hash tables grow gracefully as the number of entries increases, so that there are always less than three entries per hash bucket, on average. This allows for fast lookups regardless of the number of entries in a table.

The core provides three functions for the initialization of hash tables, Tcl_InitHashTable, Tcl_InitObjHashTable and Tcl_InitCustomHashTable.
**Tcl_InitHashTable** initializes a structure that describes a new hash table. The space for the structure is provided by the caller, not by the hash module. The value of *keyType* indicates what kinds of keys will be used for all entries in the table. All of the key types described later are allowed, with the exception of **TCL_CUSTOM_TYPE_KEYS** and **TCL_CUSTOM_PTR_KEYS**.

**Tcl_InitObjHashTable** is a wrapper around **Tcl_InitCustomHashTable** and initializes a hash table whose keys are Tcl_Obj *.

**Tcl_InitCustomHashTable** initializes a structure that describes a new hash table. The space for the structure is provided by the caller, not by the hash module. The value of *keyType* indicates what kinds of keys will be used for all entries in the table. *KeyType* must have one of the following values:

**TCL_STRING_KEYS**
Keys are null-terminated strings. They are passed to hashing routines using the address of the first character of the string.

**TCL_ONE_WORD_KEYS**
Keys are single-word values; they are passed to hashing routines and stored in hash table entries as “char *” values. The pointer value is the key; it need not (and usually does not) actually point to a string.

**TCL_CUSTOM_TYPE_KEYS**
Keys are of arbitrary type, and are stored in the entry. Hashing and comparison is determined by *typePtr*. The Tcl_HashKeyType structure is described in the section **THE TCL_HASHKEYTYPE STRUCTURE** below.

**TCL_CUSTOM_PTR_KEYS**
Keys are pointers to an arbitrary type, and are stored in the entry. Hashing and comparison is determined by *typePtr*. The Tcl_HashKeyType structure is described in the section **THE TCL_HASHKEYTYPE STRUCTURE** below.
If `keyType` is not one of the above, then it must be an integer value greater than 1. In this case the keys will be arrays of “int” values, where `keyType` gives the number of ints in each key. This allows structures to be used as keys. All keys must have the same size. Array keys are passed into hashing functions using the address of the first int in the array.

`Tcl_DeleteHashTable` deletes all of the entries in a hash table and frees up the memory associated with the table's bucket array and entries. It does not free the actual table structure (pointed to by `tablePtr`), since that memory is assumed to be managed by the client. `Tcl_DeleteHashTable` also does not free or otherwise manipulate the values of the hash table entries. If the entry values point to dynamically-allocated memory, then it is the client's responsibility to free these structures before deleting the table.

`Tcl_CreateHashEntry` locates the entry corresponding to a particular key, creating a new entry in the table if there was not already one with the given key. If an entry already existed with the given key then `*newPtr` is set to zero. If a new entry was created, then `*newPtr` is set to a non-zero value and the value of the new entry will be set to zero. The return value from `Tcl_CreateHashEntry` is a pointer to the entry, which may be used to retrieve and modify the entry's value or to delete the entry from the table.

`Tcl_DeleteHashEntry` will remove an existing entry from a table. The memory associated with the entry itself will be freed, but the client is responsible for any cleanup associated with the entry's value, such as freeing a structure that it points to.

`Tcl_FindHashEntry` is similar to `Tcl_CreateHashEntry` except that it does not create a new entry if the key doesn't exist; instead, it returns `NULL` as result.

`Tcl_GetHashValue` and `Tcl_SetHashValue` are used to read and write an entry's value, respectively. Values are stored and retrieved as type “ClientData”, which is large enough to hold a pointer value. On almost
all machines this is large enough to hold an integer value too.

**Tcl_GetHashKey** returns the key for a given hash table entry, either as a pointer to a string, a one-word (“char *”) key, or as a pointer to the first word of an array of integers, depending on the *keyType* used to create a hash table. In all cases **Tcl_GetHashKey** returns a result with type “char *”. When the key is a string or array, the result of **Tcl_GetHashKey** points to information in the table entry; this information will remain valid until the entry is deleted or its table is deleted.

**Tcl_FirstHashEntry** and **Tcl_NextHashEntry** may be used to scan all of the entries in a hash table. A structure of type “Tcl_HashSearch”, provided by the client, is used to keep track of progress through the table. **Tcl_FirstHashEntry** initializes the search record and returns the first entry in the table (or NULL if the table is empty). Each subsequent call to **Tcl_NextHashEntry** returns the next entry in the table or NULL if the end of the table has been reached. A call to **Tcl_FirstHashEntry** followed by calls to **Tcl_NextHashEntry** will return each of the entries in the table exactly once, in an arbitrary order. It is unadvisable to modify the structure of the table, e.g. by creating or deleting entries, while the search is in progress, with the exception of deleting the entry returned by **Tcl_FirstHashEntry** or **Tcl_NextHashEntry**.

**Tcl_HashStats** returns a dynamically-allocated string with overall information about a hash table, such as the number of entries it contains, the number of buckets in its hash array, and the utilization of the buckets. It is the caller’s responsibility to free the result string by passing it to *ckfree*.

The header file **tcl.h** defines the actual data structures used to implement hash tables. This is necessary so that clients can allocate Tcl_HashTable structures and so that macros can be used to read and write the values of entries. However, users of the hashing routines should never refer directly to any of the fields of any of the hash-related data structures; use the procedures and macros defined here.

---

**THE TCL_HASHKEYTYPE STRUCTURE**
Extension writers can define new hash key types by defining four procedures, initializing a `Tcl_HashKeyType` structure to describe the type, and calling `Tcl_InitCustomHashTable`. The `Tcl_HashKeyType` structure is defined as follows:

```c
typedef struct Tcl_HashKeyType {
    int version;
    int flags;
    Tcl_HashKeyProc *hashKeyProc;
    Tcl_CompareHashKeysProc *compareKeysProc;
    Tcl_AllocHashEntryProc *allocEntryProc;
    Tcl_FreeHashEntryProc *freeEntryProc;
} Tcl_HashKeyType;
```

The *version* member is the version of the table. If this structure is extended in future then the version can be used to distinguish between different structures. It should be set to `TCL_HASH_KEY_TYPE_VERSION`.

The *flags* member is 0 or one or more of the following values OR'ed together:

- **TCL_HASH_KEY_RANDOMIZE_HASH**
  There are some things, pointers for example which do not hash well because they do not use the lower bits. If this flag is set then the hash table will attempt to rectify this by randomizing the bits and then using the upper N bits as the index into the table.

- **TCL_HASH_KEY_SYSTEM_HASH**
  This flag forces Tcl to use the memory allocation procedures provided by the operating system when allocating and freeing memory used to store the hash table data structures, and not any of Tcl's own customized memory allocation routines. This is important if the hash table is to be used in the implementation of a custom set of allocation routines, or something that a custom set of allocation routines might depend on, in order to avoid any circular
The hashKeyProc member contains the address of a function called to calculate a hash value for the key.

```c
typedef unsigned int (Tcl_HashKeyProc) (Tcl_HashTable *tablePtr, void *keyPtr);
```

If this is NULL then keyPtr is used and TCL_HASH_KEY_RANDOMIZE_HASH is assumed.

The compareKeysProc member contains the address of a function called to compare two keys.

```c
typedef int (Tcl_CompareHashKeysProc) (void *keyPtr, Tcl_HashEntry *hPtr);
```

If this is NULL then the keyPtr pointers are compared. If the keys do not match then the function returns 0, otherwise it returns 1.

The allocEntryProc member contains the address of a function called to allocate space for an entry and initialize the key and clientData.

```c
typedef Tcl_HashEntry *(Tcl_AllocHashEntryProc) (Tcl_HashTable *tablePtr, void *keyPtr);
```

If this is NULL then Tcl_Alloc is used to allocate enough space for a Tcl_HashEntry, the key pointer is assigned to key.oneWordValue and the clientData is set to NULL. String keys and array keys use this function to allocate enough space for the entry and the key in one
block, rather than doing it in two blocks. This saves space for a pointer to the key from the entry and another memory allocation. Tcl_Obj* keys use this function to allocate enough space for an entry and increment the reference count on the object.

The freeEntryProc member contains the address of a function called to free space for an entry.

```c
typedef void (Tcl_FreeHashEntryProc) (Tcl_HashEntry *);
```

If this is NULL then Tcl_Free is used to free the space for the entry. Tcl_Obj* keys use this function to decrement the reference count on the object.

**KEYWORDS**

hash table, key, lookup, search, value
NAME

Tcl_GetHostName - get the name of the local host

SYNOPSIS

#include <tcl.h>
cnst char *
Tcl_GetHostName()

DESCRIPTION

Tcl_GetHostName is a utility procedure used by some of the Tcl commands. It returns a pointer to a string containing the name for the current machine, or an empty string if the name cannot be determined. The string is statically allocated, and the caller must not modify or free it.

KEYWORDS

hostname

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
Tcl_Exit, Tcl_Finalize, Tcl_CreateExitHandler,
Tcl_DeleteExitHandler, Tcl.ExitThread, Tcl_FinalizeThread,
Tcl_CreateThreadExitHandler, Tcl_DeleteThreadExitHandler,
Tcl_SetExitProc - end the application or thread (and invoke exit handlers)

SYNOPSIS
#include <tcl.h>
Tcl_Exit(status)
Tcl_Finalize()
Tcl_CreateExitHandler(proc, clientData)
Tcl_DeleteExitHandler(proc, clientData)
Tcl.ExitThread(status)
Tcl_FinalizeThread()
Tcl_CreateThreadExitHandler(proc, clientData)
Tcl_DeleteThreadExitHandler(proc, clientData)
Tcl.ExitProc *
Tcl_SetExitProc(proc)

ARGUMENTS
DESCRIPTION
KEYWORDS
Tcl.Exit(status)
Tcl_Finalize()
Tcl_CreateExitHandler(proc, clientData)
Tcl_DeleteExitHandler(proc, clientData)
Tcl.ExitThread(status)
Tcl_FinalizeThread()
Tcl_CreateThreadExitHandler(proc, clientData)
Tcl_DeleteThreadExitHandler(proc, clientData)
Tcl.ExitProc *
Tcl_SetExitProc(proc)

ARGUMENTS

- **int status (in)**: Provides information about why the application or thread exited. Exact meaning may be platform-specific. 0 usually means a normal exit, any nonzero value usually means that an error occurred.

- **Tcl.ExitProc *proc (in)**: Procedure to invoke before exiting application, or (for Tcl.SetExitProc) NULL to uninstall the current application exit procedure.

- **ClientData clientData (in)**: Arbitrary one-word value to pass to proc.

DESCRIPTION

The procedures described here provide a graceful mechanism to end the execution of a Tcl application. Exit handlers are invoked to cleanup
the application's state before ending the execution of Tcl code.

Invoke Tcl_Exit to end a Tcl application and to exit from this process. This procedure is invoked by the exit command, and can be invoked anyplace else to terminate the application. No-one should ever invoke the exit system procedure directly; always invoke Tcl_Exit instead, so that it can invoke exit handlers. Note that if other code invokes exit system procedure directly, or otherwise causes the application to terminate without calling Tcl_Exit, the exit handlers will not be run. Tcl_Exit internally invokes the exit system call, thus it never returns control to its caller. If an application exit handler has been installed (see Tcl_SetExitProc), that handler is invoked with an argument consisting of the exit status (cast to ClientData); the application exit handler should not return control to Tcl.

Tcl_Finalize is similar to Tcl_Exit except that it does not exit from the current process. It is useful for cleaning up when a process is finished using Tcl but wishes to continue executing, and when Tcl is used in a dynamically loaded extension that is about to be unloaded. On some systems Tcl is automatically notified when it is being unloaded, and it calls Tcl_Finalize internally; on these systems it not necessary for the caller to explicitly call Tcl_Finalize. However, to ensure portability, your code should always invoke Tcl_Finalize when Tcl is being unloaded, to ensure that the code will work on all platforms. Tcl_Finalize can be safely called more than once.

Tcl.ExitThread is used to terminate the current thread and invoke per-thread exit handlers. This finalization is done by Tcl_FinalizeThread, which you can call if you just want to clean up per-thread state and invoke the thread exit handlers. Tcl_Finalize calls Tcl_FinalizeThread for the current thread automatically.

Tcl_CreateExitHandler arranges for proc to be invoked by Tcl_Finalize and Tcl.Exit. Tcl_CreateThreadExitHandler arranges for proc to be invoked by Tcl_FinalizeThread and Tcl.ExitThread. This provides a hook for cleanup operations such as flushing buffers and freeing global memory. Proc should match the type Tcl.ExitProc:
typedef void Tcl_ExitProc(ClientData clientData);

The clientData parameter to proc is a copy of the clientData argument given to Tcl_CreateExitHandler or Tcl_CreateThreadExitHandler when the callback was created. Typically, clientData points to a data structure containing application-specific information about what to do in proc.

Tcl_DeleteExitHandler and Tcl_DeleteThreadExitHandler may be called to delete a previously-created exit handler. It removes the handler indicated by proc and clientData so that no call to proc will be made. If no such handler exists then Tcl_DeleteExitHandler or Tcl_DeleteThreadExitHandler does nothing.

Tcl_Finalize and Tcl_Exit execute all registered exit handlers, in reverse order from the order in which they were registered. This matches the natural order in which extensions are loaded and unloaded; if extension A loads extension B, it usually unloads B before it itself is unloaded. If extension A registers its exit handlers before loading extension B, this ensures that any exit handlers for B will be executed before the exit handlers for A.

Tcl_Finalize and Tcl_Exit call Tcl_FinalizeThread and the thread exit handlers after the process-wide exit handlers. This is because thread finalization shuts down the I/O channel system, so any attempt at I/O by the global exit handlers will vanish into the bitbucket.

Tcl_SetExitProc installs an application exit handler, returning the previously-installed application exit handler or NULL if no application handler was installed. If an application exit handler is installed, that exit handler takes over complete responsibility for finalization of Tcl's subsystems via Tcl_Finalize at an appropriate time. The argument passed to proc when it is invoked will be the exit status code (as passed to Tcl_Exit) cast to a ClientData value.

KEYWORDS
callback, cleanup, dynamic loading, end application, exit, unloading, thread
NAME
Tcl_GetIndexFromObj, Tcl_GetIndexFromObjStruct - lookup string in table of keywords

SYNOPSIS
#include <tcl.h>
int
Tcl_GetIndexFromObj(interp, objPtr, tablePtr, msg, flags, indexPtr)
int
Tcl_GetIndexFromObjStruct(interp, objPtr, structTablePtr, offset, msg, flags, indexPtr)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS

NAME
Tcl_GetIndexFromObj, Tcl_GetIndexFromObjStruct - lookup string in table of keywords

SYNOPSIS
#include <tcl.h>
int
Tcl_GetIndexFromObj(interp, objPtr, tablePtr, msg, flags, indexPtr)
int
Tcl_GetIndexFromObjStruct(interp, objPtr, structTablePtr, offset, msg, flags, indexPtr)
**ARGUMENTS**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_Interp <em>interp (in)</em></em></td>
<td>Interpreter to use for error reporting; if NULL, then no message is provided on errors.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>objPtr (in/out)</em></em></td>
<td>The string value of this object is used to search through <code>tablePtr</code>. The internal representation is modified to hold the index of the matching table entry.</td>
</tr>
<tr>
<td>**const char <strong>tablePtr (in)</strong></td>
<td>An array of null-terminated strings. The end of the array is marked by a NULL string pointer.</td>
</tr>
<tr>
<td>*<em>const void <em>structTablePtr (in)</em></em></td>
<td>An array of arbitrary type, typically some <code>struct</code> type. The first member of the structure must be a null-terminated string. The size of the structure is given by <code>offset</code>.</td>
</tr>
<tr>
<td><strong>int offset (in)</strong></td>
<td>The offset to add to <code>structTablePtr</code> to get to the next entry. The end of the array is marked by a NULL string pointer.</td>
</tr>
<tr>
<td>*<em>const char <em>msg (in)</em></em></td>
<td>Null-terminated string describing what is being looked up, such as <code>option</code>. This string is included in</td>
</tr>
</tbody>
</table>
This procedure provides an efficient way for looking up keywords, switch names, option names, and similar things where the value of an object must be one of a predefined set of values. ObjPtr is compared against each of the strings in tablePtr to find a match. A match occurs if objPtr's string value is identical to one of the strings in tablePtr, or if it is a non-empty unique abbreviation for exactly one of the strings in tablePtr and the TCL_EXACT flag was not specified; in either case the index of the matching entry is stored at *indexPtr and TCL_OK is returned.

If there is no matching entry, TCL_ERROR is returned and an error message is left in interp's result if interp is not NULL. Msg is included in the error message to indicate what was being looked up. For example, if msg is option the error message will have a form like "bad option "firt": must be first, second, or third".

If Tcl_GetIndexFromObj completes successfully it modifies the internal representation of objPtr to hold the address of the table and the index of the matching entry. If Tcl_GetIndexFromObj is invoked again with the same objPtr and tablePtr arguments (e.g. during a reinvocation of a Tcl command), it returns the matching index immediately without having to
redo the lookup operation. Note: **Tcl_GetIndexFromObj** assumes that the entries in *tablePtr* are static: they must not change between invocations. If the value of *objPtr* is the empty string, **Tcl_GetIndexFromObj** will treat it as a non-matching value and return **TCL_ERROR**.

**Tcl_GetIndexFromObjStruct** works just like **Tcl_GetIndexFromObj**, except that instead of treating *tablePtr* as an array of string pointers, it treats it as a pointer to the first string in a series of strings that have *offset* bytes between them (i.e. that there is a pointer to the first array of characters at *tablePtr*, a pointer to the second array of characters at *tablePtr*+*offset* bytes, etc.) This is particularly useful when processing things like **Tk_ConfigurationSpec**, whose string keys are in the same place in each of several array elements.

**SEE ALSO**

**Tcl_WrongNumArgs**

**KEYWORDS**

index, object, table lookup

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997 Sun Microsystems, Inc.
NAME

Tcl_CreateFileHandler, Tcl_DeleteFileHandler - associate procedure callbacks with files or devices (Unix only)

SYNOPSIS

#include <tcl.h>
Tcl_CreateFileHandler(fd, mask, proc, clientData)
Tcl_DeleteFileHandler(fd)

ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int fd (in)</td>
<td>Unix file descriptor for an open file or device.</td>
</tr>
<tr>
<td>int mask (in)</td>
<td>Conditions under which proc should be called: OR-ed combination of TCL_READABLE, TCL_WRITABLE, and TCL_EXCEPTION. May be set to 0 to temporarily disable a handler.</td>
</tr>
<tr>
<td>Tcl_FileProc *proc (in)</td>
<td>Procedure to invoke whenever the file or device indicated by file meets the conditions specified by mask.</td>
</tr>
</tbody>
</table>
DESCRIPTION

_Tcl_CreateFileHandler_ arranges for _proc_ to be invoked in the future whenever I/O becomes possible on a file or an exceptional condition exists for the file. The file is indicated by _fd_, and the conditions of interest are indicated by _mask_. For example, if _mask_ is _TCL_READABLE_, _proc_ will be called when the file is readable. The callback to _proc_ is made by _Tcl_DoOneEvent_, so _Tcl_CreateFileHandler_ is only useful in programs that dispatch events through _Tcl_DoOneEvent_ or through Tcl commands such as _vwait_.

_Proc_ should have arguments and result that match the type _Tcl_FileProc_:

```c
typedef void Tcl_FileProc(
    ClientData clientData,  
    int mask);
```

The _clientData_ parameter to _proc_ is a copy of the _clientData_ argument given to _Tcl_CreateFileHandler_ when the callback was created. Typically, _clientData_ points to a data structure containing application-specific information about the file. _Mask_ is an integer mask indicating which of the requested conditions actually exists for the file; it will contain a subset of the bits in the _mask_ argument to _Tcl_CreateFileHandler_.

There may exist only one handler for a given file at a given time. If _Tcl_CreateFileHandler_ is called when a handler already exists for _fd_, then the new callback replaces the information that was previously recorded.

_Tcl_DeleteFileHandler_ may be called to delete the file handler for _fd_; if
no handler exists for the file given by \textit{fd} then the procedure has no effect.

The purpose of file handlers is to enable an application to respond to events while waiting for files to become ready for I/O. For this to work correctly, the application may need to use non-blocking I/O operations on the files for which handlers are declared. Otherwise the application may block if it reads or writes too much data; while waiting for the I/O to complete the application will not be able to service other events. Use \texttt{Tcl\_SetChannelOption} with \texttt{-blocking} to set the channel into blocking or nonblocking mode as required.

Note that these interfaces are only supported by the Unix implementation of the Tcl notifier.

**KEYWORDS**

callback, file, handler

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME
Tcl_ListObjAppendList, Tcl_ListObjAppendElement,
Tcl_NewListObj, Tcl_SetListObj, Tcl_ListObjGetElements,
Tcl_ListObjLength, Tcl_ListObjIndex, Tcl_ListObjReplace -
manipulate Tcl objects as lists

SYNOPSIS
#include <tcl.h>
int
Tcl_ListObjAppendList(interp, listPtr, elemListPtr)
int
Tcl_ListObjAppendElement(interp, listPtr, objPtr)
Tcl_Obj *
Tcl_NewListObj(objc, objv)
Tcl_SetListObj(objPtr, objc, objv)
int
Tcl_ListObjGetElements(interp, listPtr, objcPtr, objvPtr)
int
Tcl_ListObjLength(interp, listPtr, intPtr)
int
Tcl_ListObjIndex(interp, listPtr, index, objPtrPtr)
int
Tcl_ListObjReplace(interp, listPtr, first, count, objc, objv)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS

NAME
Tcl_ListObjAppendList, Tcl_ListObjAppendElement, Tcl_NewListObj,
Tcl_SetListObj, Tcl_ListObjGetElements, Tcl_ListObjLength,
Tcl_ListObjIndex, Tcl_ListObjReplace - manipulate Tcl objects as lists

### SYNOPSIS

```c
#include <tcl.h>
int Tcl_ListObjAppendList(interp, listPtr, elemListPtr)
int Tcl_ListObjAppendElement(interp, listPtr, objPtr)
Tcl_Obj *Tcl_NewListObj(objc, objv)
Tcl_SetListObj(objPtr, objc, objv)
int Tcl_ListObjGetElements(interp, listPtr, objcPtr, objvPtr)
int Tcl_ListObjLength(interp, listPtr, intPtr)
int Tcl_ListObjIndex(interp, listPtr, index, objPtrPtr)
int Tcl_ListObjReplace(interp, listPtr, first, count, objc, objv)
```

### ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_Interp <em>interp</em></em> (in)</td>
<td>If an error occurs while converting an object to be a list object, an error message is left in the interpreter's result object unless interp is NULL.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>listPtr</em></em> (in/out)</td>
<td>Points to the list object to be manipulated. If listPtr does not already point to a list object, an attempt will be made to convert it to one.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>elemListPtr</em></em> (in/out)</td>
<td>For <strong>Tcl_ListObjAppendList</strong>, this points to a list object containing elements to be appended onto <em>listPtr</em>. Each element of <em>elemListPtr</em> will become a new element of <em>listPtr</em>. If <em>elemListPtr</em> is not NULL and does not already point to a list object, an attempt will be made to convert it to one.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>objPtr</em></em> (in)</td>
<td>For <strong>Tcl_ListObjAppendElement</strong>, points to the Tcl object that will be appended to <em>listPtr</em>. For <strong>Tcl_SetListObj</strong>, this points to the Tcl object that will be converted to a list object containing the <em>objc</em> elements of the array referenced by <em>objv</em>.</td>
</tr>
<tr>
<td>*<em>int <em>objcPtr</em></em> (in)</td>
<td>Points to location where <strong>Tcl_ListObjGetElements</strong> stores the number of element objects in <em>listPtr</em>.</td>
</tr>
<tr>
<td>**Tcl_Obj *<strong>objvPtr</strong> (out)</td>
<td>A location where <strong>Tcl_ListObjGetElements</strong> stores a pointer to an array of pointers to the element objects of <em>listPtr</em>.</td>
</tr>
<tr>
<td><strong>int objc</strong> (in)</td>
<td>The number of Tcl objects</td>
</tr>
</tbody>
</table>
That `Tcl_NewListObj` will insert into a new list object, and `Tcl_ListObjReplace` will insert into `listPtr`. For `Tcl_SetListObj`, the number of Tcl objects to insert into `objPtr`.

<table>
<thead>
<tr>
<th>*<em>Tcl_Obj <em>const objv[] (in)</em></em></th>
<th>An array of pointers to objects. <code>Tcl_NewListObj</code> will insert these objects into a new list object and <code>Tcl_ListObjReplace</code> will insert them into an existing <code>listPtr</code>. Each object will become a separate list element.</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>int <em>intPtr (out)</em></em></td>
<td>Points to location where <code>Tcl_ListObjLength</code> stores the length of the list.</td>
</tr>
<tr>
<td><strong>int index (in)</strong></td>
<td>Index of the list element that <code>Tcl_ListObjIndex</code> is to return. The first element has index 0.</td>
</tr>
<tr>
<td>**Tcl_Obj <strong>objPtrPtr (out)</strong></td>
<td>Points to place where <code>Tcl_ListObjIndex</code> is to store a pointer to the resulting list element object.</td>
</tr>
<tr>
<td><strong>int first (in)</strong></td>
<td>Index of the starting list element that <code>Tcl_ListObjReplace</code> is to replace. The list's first</td>
</tr>
</tbody>
</table>
The number of elements that Tcl_ListObjReplace is to replace.

**DESCRIPTION**

Tcl list objects have an internal representation that supports the efficient indexing and appending. The procedures described in this man page are used to create, modify, index, and append to Tcl list objects from C code.

**Tcl_ListObjAppendList** and **Tcl_ListObjAppendElement** both add one or more objects to the end of the list object referenced by *listPtr*. **Tcl_ListObjAppendList** appends each element of the list object referenced by *elemListPtr* while **Tcl_ListObjAppendElement** appends the single object referenced by *objPtr*. Both procedures will convert the object referenced by *listPtr* to a list object if necessary. If an error occurs during conversion, both procedures return **TCL_ERROR** and leave an error message in the interpreter's result object if *interp* is not NULL. Similarly, if *elemListPtr* does not already refer to a list object, **Tcl_ListObjAppendList** will attempt to convert it to one and if an error occurs during conversion, will return **TCL_ERROR** and leave an error message in the interpreter's result object if *interp* is not NULL. Both procedures invalidate any old string representation of *listPtr* and, if it was converted to a list object, free any old internal representation. Similarly, **Tcl_ListObjAppendList** frees any old internal representation of *elemListPtr* if it converts it to a list object. After appending each element in *elemListPtr*, **Tcl_ListObjAppendList** increments the element's reference count since *listPtr* now also refers to it. For the same reason, **Tcl_ListObjAppendElement** increments *objPtr*'s reference count. If no error occurs, the two procedures return **TCL_OK** after appending the objects.

**Tcl_NewListObj** and **Tcl_SetListObj** create a new object or modify an existing object to hold the *objc* elements of the array referenced by *objv*
where each element is a pointer to a Tcl object. If \textit{objc} is less than or equal to zero, they return an empty object. The new object's string representation is left invalid. The two procedures increment the reference counts of the elements in \textit{objc} since the list object now refers to them. The new list object returned by \texttt{Tcl\_NewListObj} has reference count zero.

\texttt{Tcl\_ListObjGetElements} returns a count and a pointer to an array of the elements in a list object. It returns the count by storing it in the address \textit{objcPtr}. Similarly, it returns the array pointer by storing it in the address \textit{objvPtr}. The memory pointed to is managed by Tcl and should not be freed or written to by the caller. If the list is empty, 0 is stored at \textit{objcPtr} and NULL at \textit{objvPtr}. If \textit{listPtr} is not already a list object, \texttt{Tcl\_ListObjGetElements} will attempt to convert it to one; if the conversion fails, it returns TCL\_ERROR and leaves an error message in the interpreter's result object if \textit{interp} is not NULL. Otherwise it returns TCL\_OK after storing the count and array pointer.

\texttt{Tcl\_ListObjLength} returns the number of elements in the list object referenced by \textit{listPtr}. It returns this count by storing an integer in the address \textit{intPtr}. If the object is not already a list object, \texttt{Tcl\_ListObjLength} will attempt to convert it to one; if the conversion fails, it returns TCL\_ERROR and leaves an error message in the interpreter's result object if \textit{interp} is not NULL. Otherwise it returns TCL\_OK after storing the list's length.

The procedure \texttt{Tcl\_ListObjIndex} returns a pointer to the object at element \textit{index} in the list referenced by \textit{listPtr}. It returns this object by storing a pointer to it in the address \textit{objPtrPtr}. If \textit{listPtr} does not already refer to a list object, \texttt{Tcl\_ListObjIndex} will attempt to convert it to one; if the conversion fails, it returns TCL\_ERROR and leaves an error message in the interpreter's result object if \textit{interp} is not NULL. If the index is out of range, that is, \textit{index} is negative or greater than or equal to the number of elements in the list, \texttt{Tcl\_ListObjIndex} stores a NULL in \textit{objPtrPtr} and returns TCL\_OK. Otherwise it returns TCL\_OK after storing the element's object pointer. The reference count for the list element is not incremented; the caller must do that if it needs to retain a pointer to the element.
**Tcl_ListObjReplace** replaces zero or more elements of the list referenced by *listPtr* with the *objc* objects in the array referenced by *objv*. If *listPtr* does not point to a list object, **Tcl_ListObjReplace** will attempt to convert it to one; if the conversion fails, it returns **TCL_ERROR** and leaves an error message in the interpreter's result object if *interp* is not NULL. Otherwise, it returns **TCL_OK** after replacing the objects. If *objc* is NULL, no new elements are added. If the argument *first* is zero or negative, it refers to the first element. If *first* is greater than or equal to the number of elements in the list, then no elements are deleted; the new elements are appended to the list. *count* gives the number of elements to replace. If *count* is zero or negative then no elements are deleted; the new elements are simply inserted before the one designated by *first*. **Tcl_ListObjReplace** invalidates *listPtr*’s old string representation. The reference counts of any elements inserted from *objc* are incremented since the resulting list now refers to them. Similarly, the reference counts for any replaced objects are decremented.

Because **Tcl_ListObjReplace** combines both element insertion and deletion, it can be used to implement a number of list operations. For example, the following code inserts the *objc* objects referenced by the array of object pointers *objc* just before the element *index* of the list referenced by *listPtr*:

```c
result = Tcl_ListObjReplace(interp, listPtr, index, objc, objv);
```

Similarly, the following code appends the *objc* objects referenced by the array *objc* to the end of the list *listPtr*:

```c
result = Tcl_ListObjLength(interp, listPtr, &length) if (result == TCL_OK) {
    result = Tcl_ListObjReplace(interp, listPtr, length, objc, objv);
```
The count list elements starting at first can be deleted by simply calling \texttt{Tcl\_ListObjReplace} with a NULL \texttt{objvPtr}:

\begin{verbatim}
result = Tcl_ListObjReplace(interp, listPtr, first, count, 0, NULL);
\end{verbatim}

\textbf{SEE ALSO}

\texttt{Tcl\_NewObj, Tcl\_DecrRefCount, Tcl\_IncrRefCount, Tcl\_GetObjResult}

\textbf{KEYWORDS}

append, index, insert, internal representation, length, list, list object, list type, object, object type, replace, string representation

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996-1997 Sun Microsystems, Inc.
NAME
Tcl_CreateInterp, Tcl_DeleteInterp, Tcl_InterpDeleted - create and delete Tcl command interpreters

SYNOPSIS
#include <tcl.h>
Tcl_Interp *
Tcl_CreateInterp()
Tcl_DeleteInterp(interp)
int
Tcl_InterpDeleted(interp)

ARGUMENTS

DESCRIPTION
INTERPRETERS AND MEMORY MANAGEMENT
Interpreters Passed As Arguments
Interpreter Creation And Deletion
Retrieving An Interpreter From A Data Structure

SEE ALSO

KEYWORDS

NAME
Tcl_CreateInterp, Tcl_DeleteInterp, Tcl_InterpDeleted - create and delete Tcl command interpreters

SYNOPSIS
#include <tcl.h>
Tcl_Interp *
Tcl_CreateInterp()
Tcl_DeleteInterp(interp)
int
Tcl_InterpDeleted(interp)
ARGUMENTS

Tcl_Interp *interp (in) Token for interpreter to be destroyed.

DESCRIPTION

Tcl_CreateInterp creates a new interpreter structure and returns a token for it. The token is required in calls to most other Tcl procedures, such as Tcl_CreateCommand, Tcl_Eval, and Tcl_DeleteInterp. Clients are only allowed to access a few of the fields of Tcl_Interp structures; see the Tcl_Interp and Tcl_CreateCommand man pages for details. The new interpreter is initialized with the built-in Tcl commands and with the variables documented in tclvars(n). To bind in additional commands, call Tcl_CreateCommand.

Tcl_DeleteInterp marks an interpreter as deleted; the interpreter will eventually be deleted when all calls to Tcl_Preserve for it have been matched by calls to Tcl_Release. At that time, all of the resources associated with it, including variables, procedures, and application-specific command bindings, will be deleted. After Tcl_DeleteInterp returns any attempt to use Tcl_Eval on the interpreter will fail and return TCL_ERROR. After the call to Tcl_DeleteInterp it is safe to examine the interpreter's result, query or set the values of variables, define, undefine or retrieve procedures, and examine the runtime evaluation stack. See below, in the section INTERPRETERS AND MEMORY MANAGEMENT for details.

Tcl_InterpDeleted returns nonzero if Tcl_DeleteInterp was called with interp as its argument; this indicates that the interpreter will eventually be deleted, when the last call to Tcl_Preserve for it is matched by a call to Tcl_Release. If nonzero is returned, further calls to Tcl_Eval in this interpreter will return TCL_ERROR.

Tcl_InterpDeleted is useful in deletion callbacks to distinguish between when only the memory the callback is responsible for is being deleted and when the whole interpreter is being deleted. In the former case the
callback may recreate the data being deleted, but this would lead to an infinite loop if the interpreter were being deleted.

**INTERPRETERS AND MEMORY MANAGEMENT**

*Tcl_DeleteInterp* can be called at any time on an interpreter that may be used by nested evaluations and C code in various extensions. Tcl implements a simple mechanism that allows callers to use interpreters without worrying about the interpreter being deleted in a nested call, and without requiring special code to protect the interpreter, in most cases. This mechanism ensures that nested uses of an interpreter can safely continue using it even after *Tcl_DeleteInterp* is called.

The mechanism relies on matching up calls to *Tcl_Preserve* with calls to *Tcl_Release*. If *Tcl_DeleteInterp* has been called, only when the last call to *Tcl_Preserve* is matched by a call to *Tcl_Release*, will the interpreter be freed. See the manual entry for *Tcl_Preserve* for a description of these functions.

The rules for when the user of an interpreter must call *Tcl_Preserve* and *Tcl_Release* are simple:

**Interpreters Passed As Arguments**

Functions that are passed an interpreter as an argument can safely use the interpreter without any special protection. Thus, when you write an extension consisting of new Tcl commands, no special code is needed to protect interpreters received as arguments. This covers the majority of all uses.

**Interpreter Creation And Deletion**

When a new interpreter is created and used in a call to *Tcl_Eval*, *Tcl_VarEval*, *Tcl_GlobalEval*, *Tcl_SetVar*, or *Tcl_GetVar*, a pair of calls to *Tcl_Preserve* and *Tcl_Release* should be wrapped around all uses of the interpreter. Remember that it is unsafe to use the interpreter once *Tcl_Release* has been called. To ensure that the interpreter is properly deleted when it is no longer needed, call *Tcl_InterpDeleted* to test if some other code already called *Tcl_DeleteInterp*; if not, call *Tcl_DeleteInterp* before calling...
**Tcl_Release** in your own code.

Retrieving An Interpreter From A Data Structure
When an interpreter is retrieved from a data structure (e.g. the client data of a callback) for use in **Tcl_Eval**, **Tcl_VarEval**, **Tcl_GlobalEval**, **Tcl_SetVar**, or **Tcl_GetVar**, a pair of calls to **Tcl_Preserve** and **Tcl_Release** should be wrapped around all uses of the interpreter; it is unsafe to reuse the interpreter once **Tcl_Release** has been called. If an interpreter is stored inside a callback data structure, an appropriate deletion cleanup mechanism should be set up by the code that creates the data structure so that the interpreter is removed from the data structure (e.g. by setting the field to NULL) when the interpreter is deleted. Otherwise, you may be using an interpreter that has been freed and whose memory may already have been reused.

All uses of interpreters in Tcl and Tk have already been protected. Extension writers should ensure that their code also properly protects any additional interpreters used, as described above.

**SEE ALSO**

**Tcl_Preserve**, **Tcl_Release**

**KEYWORDS**

command, create, delete, interpreter

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tcl_CreateMathFunc, Tcl_GetMathFuncInfo, Tcl_ListMathFuncs - Define, query and enumerate math functions for expressions

SYNOPSIS
#include <tcl.h>
void
Tcl_CreateMathFunc(interp, name, numArgs, argTypes, proc, clientData)
int
Tcl_GetMathFuncInfo(interp, name, numArgsPtr, argTypesPtr, procPtr, clientDataPtr)
Tcl_Obj *
Tcl_ListMathFuncs(interp, pattern)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS

NAME
Tcl_CreateMathFunc, Tcl_GetMathFuncInfo, Tcl_ListMathFuncs - Define, query and enumerate math functions for expressions

SYNOPSIS
#include <tcl.h>
void
Tcl_CreateMathFunc(interp, name, numArgs, argTypes, proc, clientData)
int
**Tcl_GetMathFuncInfo** (*interp*, *name*, *numArgsPtr*, *argTypesPtr*, *procPtr*, *clientDataPtr*)

Tcl_Obj *

**Tcl_ListMathFuncs** (*interp*, *pattern*)

### ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp (in)</em></td>
<td>Interpreter in which new function will be defined.</td>
</tr>
<tr>
<td>const char <em>name (in)</em></td>
<td>Name for new function.</td>
</tr>
<tr>
<td>int <em>numArgs (in)</em></td>
<td>Number of arguments to new function; also gives size of <em>argTypes</em> array.</td>
</tr>
<tr>
<td><strong>Tcl_ValueType</strong> <em>argTypes (in)</em></td>
<td>Points to an array giving the permissible types for each argument to function.</td>
</tr>
<tr>
<td><strong>Tcl_MathProc</strong> <em>proc (in)</em></td>
<td>Procedure that implements the function.</td>
</tr>
<tr>
<td>ClientData <em>clientData (in)</em></td>
<td>Arbitrary one-word value to pass to <em>proc</em> when it is invoked.</td>
</tr>
<tr>
<td>int <em>numArgsPtr (out)</em></td>
<td>Points to a variable that will be set to contain the number of arguments to the function.</td>
</tr>
<tr>
<td><strong>Tcl_ValueType</strong> *<em>argTypesPtr (out)</em></td>
<td>Points to a variable that will be set to contain a pointer to an array giving the permissible types for each argument to the function.</td>
</tr>
</tbody>
</table>
The `Tcl_CreateMathFunc` function is an obsolete way for applications to add additional functions to those already provided by Tcl or to replace existing functions. It should not be used by new applications, which

### DESCRIPTION

Tcl allows a number of mathematical functions to be used in expressions, such as `sin`, `cos`, and `hypot`. These functions are represented by commands in the namespace, `tcl::mathfunc`. The `Tcl_CreateMathFunc` function is an obsolete way for applications to add additional functions to those already provided by Tcl or to replace existing functions. It should not be used by new applications, which

<table>
<thead>
<tr>
<th><strong>Tcl_MathProc</strong> <code>**procPtr</code> (out)</th>
<th>Points to a variable that will be set to contain a pointer to the implementation code for the function (or NULL if the function is implemented directly in bytecode).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClientData</strong> <code>*clientDataPtr</code> (out)</td>
<td>Points to a variable that will be set to contain the clientData argument passed to <code>Tcl_CreateMathFunc</code> when the function was created if the function is not implemented directly in bytecode.</td>
</tr>
<tr>
<td><strong>const char</strong> <code>*pattern</code> (in)</td>
<td>Pattern to match against function names so as to filter them (by passing to <code>Tcl_StringMatch</code>), or NULL to not apply any filter.</td>
</tr>
</tbody>
</table>
should create math functions using `Tcl_CreateObjCommand` to create a command in the `tcl::mathfunc` namespace.

In the `Tcl_CreateMathFunc` interface, `Name` is the name of the function as it will appear in expressions. If `name` does not already exist in the `::tcl::mathfunc` namespace, then a new command is created in that namespace. If `name` does exist, then the existing function is replaced. `NumArgs` and `argTypes` describe the arguments to the function. Each entry in the `argTypes` array must be one of `TCL_INT`, `TCL_DOUBLE`, `TCL_WIDE_INT`, or `TCL_EITHER` to indicate whether the corresponding argument must be an integer, a double-precision floating value, a wide (64-bit) integer, or any, respectively.

Whenever the function is invoked in an expression Tcl will invoke `proc`. `Proc` should have arguments and result that match the type `Tcl_MathProc`:

```c
typedef int Tcl_MathProc(
    ClientData clientData,
    Tcl_Interp *interp,
    Tcl_Value *args,
    Tcl_Value *resultPtr);
```

When `proc` is invoked the `clientData` and `interp` arguments will be the same as those passed to `Tcl_CreateMathFunc`. `Args` will point to an array of `numArgs` `Tcl_Value` structures, which describe the actual arguments to the function:

```c
typedef struct Tcl_Value {
    Tcl_ValueType type;
    long intValue;
    double doubleValue;
    Tcl_WideInt wideValue;
} Tcl_Value;
```
The type field indicates the type of the argument and is one of TCL_INT, TCL_DOUBLE or TCL_WIDE_INT. It will match the argTypes value specified for the function unless the argTypes value was TCL_EITHER. Tcl converts the argument supplied in the expression to the type requested in argTypes, if that is necessary. Depending on the value of the type field, the intValue, doubleValue or wideValue field will contain the actual value of the argument.

Proc should compute its result and store it either as an integer in resultPtr->intValue or as a floating value in resultPtr->doubleValue. It should set also resultPtr->type to one of TCL_INT, TCL_DOUBLE or TCL_WIDE_INT to indicate which value was set. Under normal circumstances proc should return TCL_OK. If an error occurs while executing the function, proc should return TCL_ERROR and leave an error message in the interpreter's result.

Tcl_GetMathFuncInfo retrieves the values associated with function name that were passed to a preceding Tcl_CreateMathFunc call. Normally, the return code is TCL_OK but if the named function does not exist, TCL_ERROR is returned and an error message is placed in the interpreter's result.

If an error did not occur, the array reference placed in the variable pointed to by argTypesPtr is newly allocated, and should be released by passing it to Tcl_Free. Some functions (the standard set implemented in the core, and those defined by placing commands in the tcl::mathfunc namespace) do not have argument type information; attempting to retrieve values for them causes a NULL to be stored in the variable pointed to by procPtr and the variable pointed to by clientDataPtr will not be modified. The variable pointed to by numArgsPointer will contain -1, and no argument types will be stored in the variable pointed to by argTypesPointer.

Tcl_ListMathFuncs returns a Tcl object containing a list of all the math functions defined in the interpreter whose name matches pattern. The returned object has a reference count of zero.
expr, info, Tcl_CreateObjCommand, Tcl_Free, Tcl_NewListObj

**KEYWORDS**

expression, mathematical function
NAME

Tcl_Main, Tcl_SetMainLoop - main program and event loop definition for Tcl-based applications

SYNOPSIS

#include <tcl.h>
Tcl_Main(argc, argv, appInitProc)
Tcl_SetMainLoop(mainLoopProc)

ARGUMENTS

int argc (in)  
Number of elements in argv.

char *argv[] (in)  
Array of strings containing command-line arguments.

Tcl_AppInitProc *appInitProc (in)  
Address of an application-specific initialization procedure. The value for this argument is usually Tcl_AppInit.

Tcl_MainLoopProc *mainLoopProc (in)  
Address of an application-specific event loop procedure.

DESCRIPTION
**Tcl_Main** can serve as the main program for Tcl-based shell applications. A “shell application” is a program like tclsh or wish that supports both interactive interpretation of Tcl and evaluation of a script contained in a file given as a command line argument. **Tcl_Main** is offered as a convenience to developers of shell applications, so they do not have to reproduce all of the code for proper initialization of the Tcl library and interactive shell operation. Other styles of embedding Tcl in an application are not supported by **Tcl_Main**. Those must be achieved by calling lower level functions in the Tcl library directly. The **Tcl_Main** function has been offered by the Tcl library since release Tcl 7.4. In older releases of Tcl, the Tcl library itself defined a function *main*, but that lacks flexibility of embedding style and having a function *main* in a library (particularly a shared library) causes problems on many systems. Having *main* in the Tcl library would also make it hard to use Tcl in C++ programs, since C++ programs must have special C++ *main* functions.

Normally each shell application contains a small *main* function that does nothing but invoke **Tcl_Main**. **Tcl_Main** then does all the work of creating and running a tclsh-like application.

**Tcl_Main** is not provided by the public interface of Tcl's stub library. Programs that call **Tcl_Main** must be linked against the standard Tcl library. Extensions (stub-enabled or not) are not intended to call **Tcl_Main**.

**Tcl_Main** is not thread-safe. It should only be called by a single master thread of a multi-threaded application. This restriction is not a problem with normal use described above.

**Tcl_Main** and therefore all applications based upon it, like tclsh, use **Tcl_GetStdChannel** to initialize the standard channels to their default values. See **Tcl_StandardChannels** for more information.

**Tcl_Main** supports two modes of operation, depending on the values of *argc* and *argv*. If the first few arguments in *argv* match `?-encoding name? fileName`, where `fileName` does not begin with the character -, then `fileName` is taken to be the name of a file containing a startup script, and `name` is taken to be the name of the encoding of the
contents of that file, which **Tcl_Main** will attempt to evaluate. Otherwise, **Tcl_Main** will enter an interactive mode.

In either mode, **Tcl_Main** will define in its master interpreter the Tcl variables *argc*, *argv*, *argv0*, and *tcl_interactive*, as described in the documentation for **tclsh**.

When it has finished its own initialization, but before it processes commands, **Tcl_Main** calls the procedure given by the *appInitProc* argument. This procedure provides a “hook” for the application to perform its own initialization of the interpreter created by **Tcl_Main**, such as defining application-specific commands. The procedure must have an interface that matches the type **Tcl_AppInitProc**:

```c
typedef int Tcl_AppInitProc(Tcl_Interp *interp);
```

*AppInitProc* is almost always a pointer to **Tcl_AppInit**; for more details on this procedure, see the documentation for **Tcl_AppInit**.

When the *appInitProc* is finished, **Tcl_Main** enters one of its two modes. If a startup script has been provided, **Tcl_Main** attempts to evaluate it. Otherwise, interactive mode begins with examination of the variable *tcl_rcFileName* in the master interpreter. If that variable exists and holds the name of a readable file, the contents of that file are evaluated in the master interpreter. Then interactive operations begin, with prompts and command evaluation results written to the standard output channel, and commands read from the standard input channel and then evaluated. The prompts written to the standard output channel may be customized by defining the Tcl variables *tcl_prompt1* and *tcl_prompt2* as described in the documentation for **tclsh**. The prompts and command evaluation results are written to the standard output channel only if the Tcl variable *tcl_interactive* in the master interpreter holds a non-zero integer value.

**Tcl_SetMainLoop** allows setting an event loop procedure to be run. This allows, for example, Tk to be dynamically loaded and set its event loop. The event loop will run following the startup script. If you are in
interactive mode, setting the main loop procedure will cause the prompt to become fileevent based and then the loop procedure is called. When the loop procedure returns in interactive mode, interactive operation will continue. The main loop procedure must have an interface that matches the type `Tcl_MainLoopProc`:

```c
typedef void Tcl_MainLoopProc(void);
```

`Tcl_Main` does not return. Normally a program based on `Tcl_Main` will terminate when the `exit` command is evaluated. In interactive mode, if an EOF or channel error is encountered on the standard input channel, then `Tcl_Main` itself will evaluate the `exit` command after the main loop procedure (if any) returns. In non-interactive mode, after `Tcl_Main` evaluates the startup script, and the main loop procedure (if any) returns, `Tcl_Main` will also evaluate the `exit` command.

**SEE ALSO**

`tclsh`, `Tcl_GetStdChannel`, `Tcl_StandardChannels`, `Tcl_AppInit`, `exit`

**KEYWORDS**

`application-specific initialization`, `command-line arguments`, `main program`

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Copyright © 2000 Ajuba Solutions.
NAME
Tcl_CreateTrace, Tcl_CreateObjTrace, Tcl_DeleteTrace - arrange for command execution to be traced

SYNOPSIS
#include <tcl.h>
Tcl_Trace
Tcl_CreateTrace(interp, level, proc, clientData)
Tcl_CreateObjTrace(interp, level, flags, objProc, clientData, deleteProc)
Tcl_DeleteTrace(interp, trace)

ARGUMENTS
DESCRIPTION
KEYWORDS
**Tcl_Interp** *interp* (in)  
Interpreter containing command to be traced or untraced.

**int level** (in)  
Only commands at or below this nesting level will be traced unless 0 is specified. 1 means top-level commands only, 2 means top-level commands or those that are invoked as immediate consequences of executing top-level commands (procedure bodies, bracketed commands, etc.) and so on. A value of 0 means that commands at any level are traced.

**int flags** (in)  
Flags governing the trace execution. See below for details.

**Tcl_CmdObjTraceProc** *objProc* (in)  
Procedure to call for each command that is executed. See below for details of the calling sequence.

**Tcl_CmdTraceProc** *proc* (in)  
Procedure to call for each command that is executed. See below for details on the calling sequence.

**ClientData** clientData (in)  
Arbitrary one-word value to pass to *objProc* or *proc*. 
Tcl_CmdObjTraceDeleteProc *deleteProc (in)

Procedure to call when the trace is deleted. See below for details of the calling sequence. A NULL pointer is permissible and results in no callback when the trace is deleted.

Tcl_Trace trace (in)

Token for trace to be removed (return value from previous call to Tcl_CreateTrace).

**DESCRIPTION**

Tcl_CreateObjTrace arranges for command tracing. After it is called, objProc will be invoked before the Tcl interpreter calls any command procedure when evaluating commands in interp. The return value from Tcl_CreateObjTrace is a token for the trace, which may be passed to Tcl_DeleteTrace to remove the trace. There may be many traces in effect simultaneously for the same interpreter.

objProc should have arguments and result that match the type, Tcl_CmdObjTraceProc:

```c
typedef int Tcl_CmdObjTraceProc(
    ClientData clientData,
    Tcl_Interp* interp,
    int level,
    const char *command,
    Tcl_Command commandToken,
    int objc,
    Tcl_Obj *const objv[]);
```

The clientData and interp parameters are copies of the corresponding
arguments given to \texttt{Tcl\_CreateTrace}. \textit{ClientData} typically points to an application-specific data structure that describes what to do when \textit{objProc} is invoked. The \textit{level} parameter gives the nesting level of the command (1 for top-level commands passed to \texttt{Tcl\_Eval} by the application, 2 for the next-level commands passed to \texttt{Tcl\_Eval} as part of parsing or interpreting level-1 commands, and so on). The \textit{command} parameter points to a string containing the text of the command, before any argument substitution. The \textit{commandToken} parameter is a Tcl command token that identifies the command to be invoked. The token may be passed to \texttt{Tcl\_GetCommandName}, \texttt{Tcl\_GetCommandInfoFromToken}, or \texttt{Tcl\_SetCommandInfoFromToken} to manipulate the definition of the command. The \textit{objc} and \textit{objv} parameters designate the final parameter count and parameter vector that will be passed to the command, and have had all substitutions performed.

The \textit{objProc} callback is expected to return a standard Tcl status return code. If this code is \texttt{TCL\_OK} (the normal case), then the Tcl interpreter will invoke the command. Any other return code is treated as if the command returned that status, and the command is \textit{not} invoked.

The \textit{objProc} callback must not modify \textit{objv} in any way. It is, however, permissible to change the command by calling \texttt{Tcl\_SetCommandTokenInfo} prior to returning. Any such change takes effect immediately, and the command is invoked with the new information.

Tracing will only occur for commands at nesting level less than or equal to the \textit{level} parameter (i.e. the \textit{level} parameter to \textit{objProc} will always be less than or equal to the \textit{level} parameter to \texttt{Tcl\_CreateTrace}).

Tracing has a significant effect on runtime performance because it causes the bytecode compiler to refrain from generating in-line code for Tcl commands such as \texttt{if} and \texttt{while} in order that they may be traced. If traces for the built-in commands are not required, the \textit{flags} parameter may be set to the constant value \texttt{TCL\_ALLOW\_INLINE\_COMPILATION}. In this case, traces on built-in commands may or may not result in trace callbacks, depending on the
state of the interpreter, but run-time performance will be improved significantly. (This functionality is desirable, for example, when using \texttt{Tcl\_CreateObjTrace} to implement an execution time profiler.)

Calls to \texttt{objProc} will be made by the Tcl parser immediately before it calls the command procedure for the command (\texttt{cmdProc}). This occurs after argument parsing and substitution, so tracing for substituted commands occurs before tracing of the commands containing the substitutions. If there is a syntax error in a command, or if there is no command procedure associated with a command name, then no tracing will occur for that command. If a string passed to \texttt{Tcl\_Eval} contains multiple commands (bracketed, or on different lines) then multiple calls to \texttt{objProc} will occur, one for each command.

\texttt{Tcl\_DeleteTrace} removes a trace, so that no future calls will be made to the procedure associated with the trace. After \texttt{Tcl\_DeleteTrace} returns, the caller should never again use the \textit{trace} token.

When \texttt{Tcl\_DeleteTrace} is called, the interpreter invokes the \textit{deleteProc} that was passed as a parameter to \texttt{Tcl\_CreateObjTrace}. The \textit{deleteProc} must match the type, \texttt{Tcl\_CmdObjTraceDeleteProc}:

\begin{verbatim}
typedef void Tcl_CmdObjTraceDeleteProc(
    ClientData clientData);
\end{verbatim}

The \textit{clientData} parameter will be the same as the \textit{clientData} parameter that was originally passed to \texttt{Tcl\_CreateObjTrace}.

\texttt{Tcl\_CreateTrace} is an alternative interface for command tracing, \textit{not recommended for new applications}. It is provided for backward compatibility with code that was developed for older versions of the Tcl interpreter. It is similar to \texttt{Tcl\_CreateObjTrace}, except that its \textit{proc} parameter should have arguments and result that match the type \texttt{Tcl\_CmdTraceProc}:

\begin{verbatim}
typedef void Tcl_CmdTraceProc(
    ...
    ));
\end{verbatim}
ClientData clientData,
Tcl_Interp *interp,
int level,
char *command,
Tcl_CmdProc *cmdProc,
ClientData cmdClientData,
int argc,
const char *argv[];

The parameters to the proc callback are similar to those of the objProc callback above. The commandToken is replaced with cmdProc, a pointer to the (string-based) command procedure that will be invoked; and cmdClientData, the client data that will be passed to the procedure. The objc parameter is replaced with an argv parameter, that gives the arguments to the command as character strings. Proc must not modify the command or argv strings.

If a trace created with Tcl_CreateTrace is in effect, inline compilation of Tcl commands such as if and while is always disabled. There is no notification when a trace created with Tcl_CreateTrace is deleted. There is no way to be notified when the trace created by Tcl_CreateTrace is deleted. There is no way for the proc associated with a call to Tcl_CreateTrace to abort execution of command.

KEYWORDS

command, create, delete, interpreter, trace
NAME
Tcl_FindExecutable, Tcl_GetNameOfExecutable - identify or return the name of the binary file containing the application

SYNOPSIS
#include <tcl.h>
void
Tcl_FindExecutable(argv0)
const char *
Tcl_GetNameOfExecutable()

ARGUMENTS
DESCRIPTION
KEYWORDS

NAME

Tcl_FindExecutable, Tcl_GetNameOfExecutable - identify or return the name of the binary file containing the application

SYNOPSIS
#include <tcl.h>
void
Tcl_FindExecutable(argv0)
const char *
Tcl_GetNameOfExecutable()

ARGUMENTS
char *argv0 (in) The first command-line argument to the program, which gives the application's name.
**DESCRIPTION**

The **Tcl_FindExecutable** procedure computes the full path name of the executable file from which the application was invoked and saves it for Tcl's internal use. The executable's path name is needed for several purposes in Tcl. For example, it is needed on some platforms in the implementation of the **load** command. It is also returned by the **info nameofexecutable** command.

On UNIX platforms this procedure is typically invoked as the very first thing in the application's main program; it must be passed `argv[0]` as its argument. It is important not to change the working directory before the invocation. **Tcl_FindExecutable** uses `argv0` along with the **PATH** environment variable to find the application's executable, if possible. If it fails to find the binary, then future calls to **info nameofexecutable** will return an empty string.

**Tcl_GetNameOfExecutable** simply returns a pointer to the internal full path name of the executable file as computed by **Tcl_FindExecutable**. This procedure call is the C API equivalent to the **info nameofexecutable** command. NULL is returned if the internal full path name has not been computed or unknown.

**KEYWORDS**

*binary, executable file*
NAME

Tcl_CreateTimerHandler, Tcl_DeleteTimerHandler - call a procedure at a given time

SYNOPSIS

```
#include <tcl.h>
Tcl_TimerToken
Tcl_CreateTimerHandler(milliseconds, proc, clientData)
Tcl_DeleteTimerHandler(token)
```

ARGUMENTS

- **int milliseconds (in)**: How many milliseconds to wait before invoking `proc`.
- **Tcl_TimerProc *proc (in)**: Procedure to invoke after `milliseconds` have elapsed.
- **ClientData clientData (in)**: Arbitrary one-word value to pass to `proc`.
- **Tcl_TimerToken token (in)**: Token for previously created timer handler (the return value from some previous call to `Tcl_CreateTimerHandler`).

DESCRIPTION
**Tcl_CreateTimerHandler** arranges for *proc* to be invoked at a time *milliseconds* milliseconds in the future. The callback to *proc* will be made by **Tcl_DoOneEvent**, so **Tcl_CreateTimerHandler** is only useful in programs that dispatch events through **Tcl_DoOneEvent** or through Tcl commands such as **vwait**. The call to *proc* may not be made at the exact time given by *milliseconds*: it will be made at the next opportunity after that time. For example, if **Tcl_DoOneEvent** is not called until long after the time has elapsed, or if there are other pending events to process before the call to *proc*, then the call to *proc* will be delayed.

*Proc* should have arguments and return value that match the type **Tcl_TimerProc**:

```c
typedef void Tcl_TimerProc(ClientData clientData);
```

The *clientData* parameter to *proc* is a copy of the *clientData* argument given to **Tcl_CreateTimerHandler** when the callback was created. Typically, *clientData* points to a data structure containing application-specific information about what to do in *proc*.

**Tcl_DeleteTimerHandler** may be called to delete a previously created timer handler. It deletes the handler indicated by *token* so that no call to *proc* will be made; if that handler no longer exists (e.g. because the time period has already elapsed and *proc* has been invoked then **Tcl_DeleteTimerHandler** does nothing. The tokens returned by **Tcl_CreateTimerHandler** never have a value of NULL, so if NULL is passed to **Tcl_DeleteTimerHandler** then the procedure does nothing.

**KEYWORDS**

callback, clock, handler, timer
**NAME**

Tcl_SetRecursionLimit - set maximum allowable nesting depth in interpreter

**SYNOPSIS**

```c
#include <tcl.h>
int Tcl_SetRecursionLimit(interp, depth)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interp *interp</code> (in)</td>
<td>Interpreter whose recursion limit is to be set. Must be greater than zero.</td>
</tr>
<tr>
<td><code>int depth</code> (in)</td>
<td>New limit for nested calls to <code>Tcl_Eval</code> for <code>interp</code>.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

At any given time Tcl enforces a limit on the number of recursive calls that may be active for `Tcl_Eval` and related procedures such as `Tcl_GlobalEval`. Any call to `Tcl_Eval` that exceeds this depth is aborted with an error. By default the recursion limit is 1000.

`Tcl_SetRecursionLimit` may be used to change the maximum allowable nesting depth for an interpreter. The `depth` argument specifies a new limit for `interp`, and `Tcl_SetRecursionLimit` returns the old limit. To read out the old limit without modifying it, invoke...
Tcl_SetRecursionLimit with depth equal to 0.

The Tcl_SetRecursionLimit only sets the size of the Tcl call stack: it cannot by itself prevent stack overflows on the C stack being used by the application. If your machine has a limit on the size of the C stack, you may get stack overflows before reaching the limit set by Tcl_SetRecursionLimit. If this happens, see if there is a mechanism in your system for increasing the maximum size of the C stack.

**KEYWORDS**

nesting depth, recursion

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1989-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Tcl_GetOpenFile - Return a FILE* for a channel registered in the given interpreter (Unix only)

**SYNOPSIS**

```
#include <tcl.h>
int
Tcl_GetOpenFile(interp, chanID, write, checkUsage, filePtr)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th><strong>Tcl_Interp</strong> <em>interp</em> (in)</th>
<th>Tcl interpreter from which file handle is to be obtained.</th>
</tr>
</thead>
<tbody>
<tr>
<td>const char *chanID (in)</td>
<td>String identifying channel, such as <strong>stdin</strong> or <strong>file4</strong>.</td>
</tr>
<tr>
<td>int write (in)</td>
<td>Non-zero means the file will be used for writing, zero means it will be used for reading.</td>
</tr>
<tr>
<td>int checkUsage (in)</td>
<td>If non-zero, then an error will be generated if the file was not opened for the access indicated by <strong>write</strong>.</td>
</tr>
<tr>
<td>ClientData *filePtr (out)</td>
<td>Points to word in which to</td>
</tr>
</tbody>
</table>
DESCRIPTION

Tcl_GetOpenFile takes as argument a file identifier of the form returned by the open command and returns at *filePtr a pointer to the FILE structure for the file. The write argument indicates whether the FILE pointer will be used for reading or writing. In some cases, such as a channel that connects to a pipeline of subprocesses, different FILE pointers will be returned for reading and writing. Tcl_GetOpenFile normally returns TCL_OK. If an error occurs in Tcl_GetOpenFile (e.g. chanID did not make any sense or checkUsage was set and the file was not opened for the access specified by write) then TCL_ERROR is returned and the interpreter's result will contain an error message. In the current implementation checkUsage is ignored and consistency checks are always performed.

Note that this interface is only supported on the Unix platform.

KEYWORDS

channel, file handle, permissions, pipeline, read, write
NAME
Tcl_SplitPath, Tcl_JoinPath, Tcl_GetPathType - manipulate platform-dependent file paths

SYNOPSIS
#include <tcl.h>
Tcl_SplitPath(path, argcPtr, argvPtr)
char *
Tcl_JoinPath(argc, argv, resultPtr)
Tcl_PathType
Tcl_GetPathType(path)

ARGUMENTS
DESCRIPTION
KEYWORDS

const char *path (in) File path in a form appropriate for the current
platform (see the filename manual entry for acceptable forms for path names).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *argcPtr (out)</td>
<td>Filled in with number of path elements in <em>path</em>.</td>
</tr>
<tr>
<td>const char ***argvPtr (out)</td>
<td>*argvPtr will be filled in with the address of an array of pointers to the strings that are the extracted elements of <em>path</em>. There will be *argcPtr valid entries in the array, followed by a NULL entry.</td>
</tr>
<tr>
<td>int argc (in)</td>
<td>Number of elements in <em>argv</em>.</td>
</tr>
<tr>
<td>const char *const *argv (in)</td>
<td>Array of path elements to merge together into a single path.</td>
</tr>
<tr>
<td>Tcl_DString *resultPtr (in/out)</td>
<td>A pointer to an initialized Tcl_DString to which the result of Tcl_JoinPath will be appended.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

These procedures have been superceded by the objectified procedures in the FileSystem man page, which are more efficient.

These procedures may be used to disassemble and reassemble file paths in a platform independent manner: they provide C-level access to
the same functionality as the **file split**, **file join**, and **file pathtype** commands.

**Tcl_SplitPath** breaks a path into its constituent elements, returning an array of pointers to the elements using *argcPtr* and *argvPtr*. The area of memory pointed to by *argvPtr* is dynamically allocated; in addition to the array of pointers, it also holds copies of all the path elements. It is the caller's responsibility to free all of this storage. For example, suppose that you have called **Tcl_SplitPath** with the following code:

```c
int argc;
char *path;
char **argv;
...
Tcl_SplitPath(string, &argc, &argv);
```

Then you should eventually free the storage with a call like the following:

```c
Tcl_Free((char *) argv);
```

**Tcl_JoinPath** is the inverse of **Tcl_SplitPath**: it takes a collection of path elements given by *argc* and *argv* and generates a result string that is a properly constructed path. The result string is appended to *resultPtr*. *ResultPtr* must refer to an initialized **Tcl_DString**.

If the result of **Tcl_SplitPath** is passed to **Tcl_JoinPath**, the result will refer to the same location, but may not be in the same form. This is because **Tcl_SplitPath** and **Tcl_JoinPath** eliminate duplicate path separators and return a normalized form for each platform.

**Tcl_GetPathType** returns the type of the specified *path*, where **Tcl_PathType** is one of **TCL_PATH_ABSOLUTE**, **TCL_PATH_RELATIVE**, or **TCL_PATH_VOLUME_RELATIVE**. See the **filename** manual entry for a description of the path types for each
platform.

**KEYWORDS**

file, filename, join, path, split, type

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
NAME
Tcl_NewObj, Tcl_DuplicateObj, Tcl_IncrRefCount,
Tcl_DecrRefCount, Tcl_IsShared, Tcl_InvalidateStringRep -
manipulate Tcl objects

SYNOPSIS
#include <tcl.h>
Tcl_Obj *
Tcl_NewObj()
Tcl_Obj *
Tcl_DuplicateObj(objPtr)
Tcl_IncrRefCount(objPtr)
Tcl_DecrRefCount(objPtr)

int
Tcl_IsShared(objPtr)
Tcl_InvalidateStringRep(objPtr)

ARGUMENTS
INTRODUCTION
THE TCL_OBJ STRUCTURE
EXAMPLE OF THE LIFETIME OF AN OBJECT
STORAGE MANAGEMENT OF OBJECTS
SEE ALSO
KEYWORDS
**Tcl**'s *dual-ported* objects provide a general-purpose mechanism for storing and exchanging Tcl values. They largely replace the use of strings in Tcl. For example, they are used to store variable values, command arguments, command results, and scripts. Tcl objects behave like strings but also hold an internal representation that can be manipulated more efficiently. For example, a Tcl list is now represented as an object that holds the list's string representation as well as an array of pointers to the objects for each list element. Dual-ported objects avoid most runtime type conversions. They also improve the speed of many operations since an appropriate representation is
immediately available. The compiler itself uses Tcl objects to cache the instruction bytecodes resulting from compiling scripts.

The two representations are a cache of each other and are computed lazily. That is, each representation is only computed when necessary, it is computed from the other representation, and, once computed, it is saved. In addition, a change in one representation invalidates the other one. As an example, a Tcl program doing integer calculations can operate directly on a variable's internal machine integer representation without having to constantly convert between integers and strings. Only when it needs a string representing the variable's value, say to print it, will the program regenerate the string representation from the integer. Although objects contain an internal representation, their semantics are defined in terms of strings: an up-to-date string can always be obtained, and any change to the object will be reflected in that string when the object's string representation is fetched. Because of this representation invalidation and regeneration, it is dangerous for extension writers to access Tcl_Obj fields directly. It is better to access Tcl_Obj information using procedures like Tcl_GetStringFromObj and Tcl_GetString.

Objects are allocated on the heap and are referenced using a pointer to their Tcl_Obj structure. Objects are shared as much as possible. This significantly reduces storage requirements because some objects such as long lists are very large. Also, most Tcl values are only read and never modified. This is especially true for procedure arguments, which can be shared between the caller and the called procedure. Assignment and argument binding is done by simply assigning a pointer to the value. Reference counting is used to determine when it is safe to reclaim an object's storage.

Tcl objects are typed. An object's internal representation is controlled by its type. Several types are predefined in the Tcl core including integer, double, list, and bytecode. Extension writers can extend the set of types by defining their own Tcl_ObjType structs.

**THE TCL_OBJC STRUCTURE**

Each Tcl object is represented by a Tcl_Obj structure which is defined
as follows.

```c
typedef struct Tcl_Obj {
    int refCount;
    char *bytes;
    int length;
    Tcl_ObjType *typePtr;
    union {
        long longValue;
        double doubleValue;
        void *otherValuePtr;
        Tcl_WideInt wideValue;
        struct {
            void *ptr1;
            void *ptr2;
        } twoPtrValue;
        struct {
            void *ptr;
            unsigned long value;
        } ptrAndLongRep;
    } internalRep;
} Tcl_Obj;
```

The `bytes` and the `length` members together hold an object's UTF-8 string representation, which is a counted string not containing null bytes (UTF-8 null characters should be encoded as a two byte sequence: 192, 128.) `bytes` points to the first byte of the string representation. The `length` member gives the number of bytes. The byte array must always have a null byte after the last data byte, at offset `length`; this allows string representations to be treated as conventional null-terminated C strings. C programs use `Tcl_GetStringFromObj` and `Tcl_GetString` to get an object's string representation. If `bytes` is NULL, the string representation is invalid.

An object's type manages its internal representation. The member `typePtr` points to the Tcl_ObjType structure that describes the type. If
typePtr is NULL, the internal representation is invalid.

The internalRep union member holds an object's internal representation. This is either a (long) integer, a double-precision floating-point number, a pointer to a value containing additional information needed by the object's type to represent the object, a Tcl_WideInt integer, two arbitrary pointers, or a pair made up of an unsigned long integer and a pointer.

The refCount member is used to tell when it is safe to free an object's storage. It holds the count of active references to the object. Maintaining the correct reference count is a key responsibility of extension writers. Reference counting is discussed below in the section STORAGE MANAGEMENT OF OBJECTS.

Although extension writers can directly access the members of a Tcl_Obj structure, it is much better to use the appropriate procedures and macros. For example, extension writers should never read or update refCount directly; they should use macros such as Tcl_IncrRefCount and Tcl_IsShared instead.

A key property of Tcl objects is that they hold two representations. An object typically starts out containing only a string representation: it is untyped and has a NULL typePtr. An object containing an empty string or a copy of a specified string is created using Tcl_NewObj or Tcl_NewStringObj respectively. An object's string value is gotten with Tcl_GetStringFromObj or Tcl_GetString and changed with Tcl_SetStringObj. If the object is later passed to a procedure like Tcl_GetIntFromObj that requires a specific internal representation, the procedure will create one and set the object's typePtr. The internal representation is computed from the string representation. An object's two representations are duals of each other: changes made to one are reflected in the other. For example, Tcl_ListObjReplace will modify an object's internal representation and the next call to Tcl_GetStringFromObj or Tcl_GetString will reflect that change.

Representations are recomputed lazily for efficiency. A change to one representation made by a procedure such as Tcl_ListObjReplace is
not reflected immediately in the other representation. Instead, the other representation is marked invalid so that it is only regenerated if it is needed later. Most C programmers never have to be concerned with how this is done and simply use procedures such as `Tcl_GetBooleanFromObj` or `Tcl_ListObjIndex`. Programmers that implement their own object types must check for invalid representations and mark representations invalid when necessary. The procedure `Tcl_InvalidateStringRep` is used to mark an object's string representation invalid and to free any storage associated with the old string representation.

Objects usually remain one type over their life, but occasionally an object must be converted from one type to another. For example, a C program might build up a string in an object with repeated calls to `Tcl_AppendToObj`, and then call `Tcl_ListObjIndex` to extract a list element from the object. The same object holding the same string value can have several different internal representations at different times. Extension writers can also force an object to be converted from one type to another using the `Tcl_ConvertToType` procedure. Only programmers that create new object types need to be concerned about how this is done. A procedure defined as part of the object type's implementation creates a new internal representation for an object and changes its `typePtr`. See the man page for `Tcl_RegisterObjType` to see how to create a new object type.

## EXAMPLE OF THE LIFETIME OF AN OBJECT

As an example of the lifetime of an object, consider the following sequence of commands:

```bash
set x 123
```

This assigns to `x` an untyped object whose `bytes` member points to `123` and `length` member contains 3. The object's `typePtr` member is NULL.
puts "x is $x"

x's string representation is valid (since bytes is non-NULL) and is fetched for the command.

incr x

The incr command first gets an integer from x's object by calling Tcl_GetIntFromObj. This procedure checks whether the object is already an integer object. Since it is not, it converts the object by setting the object's internalRep.longValue member to the integer 123 and setting the object's typePtr to point to the integer Tcl_ObjType structure. Both representations are now valid. incr increments the object's integer internal representation then invalidates its string representation (by calling Tcl_InvalidateStringRep) since the string representation no longer corresponds to the internal representation.

puts "x is now $x"

The string representation of x's object is needed and is recomputed. The string representation is now 124 and both representations are again valid.

STORAGE MANAGEMENT OF OBJECTS

Tcl objects are allocated on the heap and are shared as much as possible to reduce storage requirements. Reference counting is used to determine when an object is no longer needed and can safely be freed. An object just created by Tcl_NewObj or Tcl_NewStringObj has refCount 0. The macro Tcl_IncrRefCount increments the reference count when a new reference to the object is created. The macro Tcl_DecrRefCount decrements the count when a reference is no longer needed and, if the object's reference count drops to zero, frees
its storage. An object shared by different code or data structures has refCount greater than 1. Incrementing an object's reference count ensures that it will not be freed too early or have its value change accidentally.

As an example, the bytecode interpreter shares argument objects between calling and called Tcl procedures to avoid having to copy objects. It assigns the call's argument objects to the procedure's formal parameter variables. In doing so, it calls Tcl_IncrRefCount to increment the reference count of each argument since there is now a new reference to it from the formal parameter. When the called procedure returns, the interpreter calls Tcl_DecrRefCount to decrement each argument's reference count. When an object's reference count drops less than or equal to zero, Tcl_DecrRefCount reclaims its storage. Most command procedures do not have to be concerned about reference counting since they use an object's value immediately and do not retain a pointer to the object after they return. However, if they do retain a pointer to an object in a data structure, they must be careful to increment its reference count since the retained pointer is a new reference.

Command procedures that directly modify objects such as those for lappend and linsert must be careful to copy a shared object before changing it. They must first check whether the object is shared by calling Tcl_IsShared. If the object is shared they must copy the object by using Tcl_DuplicateObj; this returns a new duplicate of the original object that has refCount 0. If the object is not shared, the command procedure "owns" the object and can safely modify it directly. For example, the following code appears in the command procedure that implements linsert. This procedure modifies the list object passed to it in objv[1] by inserting objc-3 new elements before index.

```c
listPtr = objv[1];
if (Tcl_IsShared(listPtr)) {
    listPtr = Tcl_DuplicateObj(listPtr);
}
result = Tcl_ListObjReplace(interp, listPtr, index,
```
As another example, `incr`'s command procedure must check whether the variable's object is shared before incrementing the integer in its internal representation. If it is shared, it needs to duplicate the object in order to avoid accidentally changing values in other data structures.

SEE ALSO

Tcl_ConvertToType, Tcl_GetIntFromObj, Tcl_ListObjAppendElement, Tcl_ListObjIndex, Tcl_ListObjReplace, Tcl_RegisterObjType

KEYWORDS

internal representation, object, object creation, object type, reference counting, string representation, type conversion

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996-1997 Sun Microsystems, Inc.
NAME

Tcl_GetStdChannel, Tcl_SetStdChannel - procedures for retrieving and replacing the standard channels

SYNOPSIS

#include <tcl.h>
Tcl_Channel
Tcl_GetStdChannel(type)
Tcl_SetStdChannel(channel, type)

ARGUMENTS

DESCRIPTION

SEE ALSO

KEYWORDS

NAME

Tcl_GetStdChannel, Tcl_SetStdChannel - procedures for retrieving and replacing the standard channels

SYNOPSIS

#include <tcl.h>
Tcl_Channel
Tcl_GetStdChannel(type)
Tcl_SetStdChannel(channel, type)

ARGUMENTS

int type (in)

The identifier for the standard channel to retrieve or modify. Must be one of TCL_STDIN, TCL_STDOUT, or
**TCL_STDERR.**

Tcl_Channel **channel** (in)

The channel to use as the new value for the specified standard channel.

**DESCRIPTION**

Tcl defines three special channels that are used by various I/O related commands if no other channels are specified. The standard input channel has a channel name of stdin and is used by read and gets. The standard output channel is named stdout and is used by puts. The standard error channel is named stderr and is used for reporting errors. In addition, the standard channels are inherited by any child processes created using exec or open in the absence of any other redirections.

The standard channels are actually aliases for other normal channels. The current channel associated with a standard channel can be retrieved by calling **Tcl_GetStdChannel** with one of TCL_STDIN, TCL_STDOUT, or TCL_STDERR as the type. The return value will be a valid channel, or NULL.

A new channel can be set for the standard channel specified by type by calling **Tcl_SetStdChannel** with a new channel or NULL in the channel argument. If the specified channel is closed by a later call to **Tcl_Close**, then the corresponding standard channel will automatically be set to NULL.

If a non-NULL value for channel is passed to **Tcl_SetStdChannel**, then that same value should be passed to **Tcl_RegisterChannel**, like so:

```
Tcl_RegisterChannel(NULL, channel);
```

This is a workaround for a misfeature in **Tcl_SetStdChannel** that it fails to do some reference counting housekeeping. This misfeature cannot
be corrected without contradicting the assumptions of some existing code that calls \texttt{Tcl\_SetStdChannel}.

If \texttt{Tcl\_GetStdChannel} is called before \texttt{Tcl\_SetStdChannel}, Tcl will construct a new channel to wrap the appropriate platform-specific standard file handle. If \texttt{Tcl\_SetStdChannel} is called before \texttt{Tcl\_GetStdChannel}, then the default channel will not be created.

If one of the standard channels is set to NULL, either by calling \texttt{Tcl\_SetStdChannel} with a NULL \texttt{channel} argument, or by calling \texttt{Tcl\_Close} on the channel, then the next call to \texttt{Tcl\_CreateChannel} will automatically set the standard channel with the newly created channel. If more than one standard channel is NULL, then the standard channels will be assigned starting with standard input, followed by standard output, with standard error being last.

See \texttt{Tcl\_StandardChannels} for a general treatise about standard channels and the behaviour of the Tcl library with regard to them.

\begin{tabular}{|l|}
\hline
\textbf{SEE ALSO} \\
\hline
\texttt{Tcl\_Close, Tcl\_CreateChannel, Tcl\_Main, tclsh} \\
\hline
\textbf{KEYWORDS} \\
\hline
standard channel, standard input, standard output, standard error \\
\hline
\end{tabular}
NAME
Tcl_StackChannel, Tcl_UnstackChannel,
Tcl_GetStackedChannel, Tcl_GetTopChannel - manipulate
stacked I/O channels

SYNOPSIS
#include <tcl.h>
Tcl_Channel
Tcl_StackChannel(interp, typePtr, clientData, mask, channel)
int
Tcl_UnstackChannel(interp, channel)
Tcl_Channel
Tcl_GetStackedChannel(channel)
Tcl_Channel
Tcl_GetTopChannel(channel)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS

NAME
Tcl_StackChannel, Tcl_UnstackChannel, Tcl_GetStackedChannel,
Tcl_GetTopChannel - manipulate stacked I/O channels

SYNOPSIS
#include <tcl.h>
Tcl_Channel
Tcl_StackChannel(interp, typePtr, clientData, mask, channel)
int
Tcl_UnstackChannel(interp, channel)
Tcl_Channel
**Tcl_GetStackedChannel(channel)**

Tcl_Channel

**Tcl_GetTopChannel(channel)**

### ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp * interp (in)</strong></td>
<td>Interpreter for error reporting.</td>
</tr>
<tr>
<td><strong>Tcl_ChannelType * typePtr (in)</strong></td>
<td>The new channel I/O procedures to use for <em>channel</em>.</td>
</tr>
<tr>
<td><strong>ClientData clientData (in)</strong></td>
<td>Arbitrary one-word value to pass to channel I/O procedures.</td>
</tr>
<tr>
<td><strong>int mask (in)</strong></td>
<td>Conditions under which <em>channel</em> will be used: OR-ed combination of <em>TCL_READABLE</em>, <em>TCL_WRITABLE</em> and <em>TCL_EXCEPTION</em>. This can be a subset of the operations currently allowed on <em>channel</em>.</td>
</tr>
<tr>
<td><strong>Tcl_Channel channel (in)</strong></td>
<td>An existing Tcl channel such as returned by <em>Tcl_CreateChannel</em>.</td>
</tr>
</tbody>
</table>

### DESCRIPTION

These functions are for use by extensions that add processing layers to Tcl I/O channels. Examples include compression and encryption modules. These functions transparently stack and unstack a new
channel on top of an existing one. Any number of channels can be stacked together.

The implementation of the Tcl channel code was rewritten in 8.3.2 to correct some problems with the previous implementation with regard to stacked channels. Anyone using stacked channels or creating stacked channel drivers should update to the new

**TCL_CHANNEL_VERSION_2** Tcl_ChannelType structure. See Tcl_CreateChannel for details.

Tcl_StackChannel stacks a new channel on an existing channel with the same name that was registered for channel by Tcl_RegisterChannel.

Tcl_StackChannel works by creating a new channel structure and placing itself on top of the channel stack. EOL translation, encoding and buffering options are shared between all channels in the stack. The hidden channel does no buffering, newline translations, or character set encoding. Instead, the buffering, newline translations, and encoding functions all remain at the top of the channel stack. A pointer to the new top channel structure is returned. If an error occurs when stacking the channel, NULL is returned instead.

The *mask* parameter specifies the operations that are allowed on the new channel. These can be a subset of the operations allowed on the original channel. For example, a read-write channel may become read-only after the Tcl_StackChannel call.

Closing a channel closes the channels stacked below it. The close of stacked channels is executed in a way that allows buffered data to be properly flushed.

Tcl_UnstackChannel reverses the process. The old channel is associated with the channel name, and the processing module added by Tcl_StackChannel is destroyed. If there is no old channel, then Tcl_UnstackChannel is equivalent to Tcl_Close. If an error occurs unstacking the channel, TCL_ERROR is returned, otherwise TCL_OK is returned.
**Tcl_GetTopChannel** returns the top channel in the stack of channels the supplied channel is part of.

**Tcl_GetStackedChannel** returns the channel in the stack of channels which is just below the supplied channel.

---

**SEE ALSO**

Notifier, **Tcl_CreateChannel**, **Tcl_OpenFileChannel**, `vwait(n)`.

---

**KEYWORDS**

channel, compression

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1999-2000 Ajuba Solutions.
NAME

Tcl_SignalId, Tcl_SignalMsg - Convert signal codes

SYNOPSIS

#include <tcl.h>
const char *
Tcl_SignalId(sig)
const char *
Tcl_SignalMsg(sig)

ARGUMENTS

int sig (in) A POSIX signal number such as SIGPIPE.

DESCRIPTION

Tcl_SignalId and Tcl_SignalMsg return a string representation of the provided signal number (sig). Tcl_SignalId returns a machine-readable textual identifier such as “SIGPIPE”. Tcl_SignalMsg returns a human-readable string such as “bus error”. The strings returned by these functions are statically allocated and the caller must not free or modify them.

KEYWORDS

signals, signal numbers

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

Tcl_Sleep - delay execution for a given number of milliseconds

SYNOPSIS

#include <tcl.h>
Tcl_Sleep(ms)

ARGUMENTS

int ms (in)  Number of milliseconds to sleep.

DESCRIPTION

This procedure delays the calling process by the number of milliseconds given by the ms parameter and returns after that time has elapsed. It is typically used for things like flashing a button, where the delay is short and the application need not do anything while it waits. For longer delays where the application needs to respond to other events during the delay, the procedure Tcl_CreateTimerHandler should be used instead of Tcl_Sleep.

KEYWORDS

sleep, time, wait

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tcl_SourceRCFile - source the Tcl rc file

SYNOPSIS

#include <tcl.h>

void

Tcl_SourceRCFile(interp)

ARGUMENTS

Tcl_Interp *interp (in) Tcl interpreter to source rc file into.

DESCRIPTION

Tcl_SourceRCFile is used to source the Tcl rc file at startup. It is typically invoked by Tcl_Main or Tk_Main. The name of the file sourced is obtained from the global variable tcl_rcFileName in the interpreter given by interp. If this variable is not defined, or if the file it indicates cannot be found, no action is taken.

KEYWORDS

application-specific initialization, main program, rc file
NAME
    Tcl_NewDictObj, Tcl_DictObjPut, Tcl_DictObjGet,
    Tcl_DictObjRemove, Tcl_DictObjSize, Tcl_DictObjFirst,
    Tcl_DictObjNext, Tcl_DictObjDone, Tcl_DictObjPutKeyList,
    Tcl_DictObjRemoveKeyList - manipulate Tcl objects as
dictionaries
SYNOPSIS
    #include <tcl.h>
    Tcl_Obj *
    Tcl_NewDictObj()
    int
    Tcl_DictObjGet(interp, dictPtr, keyPtr, valuePtrPtr)
    int
    Tcl_DictObjPut(interp, dictPtr, keyPtr, valuePtr)
    int
    Tcl_DictObjRemove(interp, dictPtr, keyPtr)
    int
    Tcl_DictObjSize(interp, dictPtr, sizePtr)
    int
    Tcl_DictObjFirst(interp, dictPtr, searchPtr,
    keyPtrPtr, valuePtrPtr, donePtr)
    void
    Tcl_DictObjNext(searchPtr, keyPtrPtr, valuePtrPtr, donePtr)
    void
    Tcl_DictObjDone(searchPtr)
    int
    Tcl_DictObjPutKeyList(interp, dictPtr, keyc, keyv, valuePtr)
    int
    Tcl_DictObjRemoveKeyList(interp, dictPtr, keyc, keyv)
ARGUMENTS
DESCRIPTION
NAME

Tcl_NewDictObj, Tcl_DictObjPut, Tcl_DictObjGet, Tcl_DictObjRemove, Tcl_DictObjSize, Tcl_DictObjFirst, Tcl_DictObjNext, Tcl_DictObjDone, Tcl_DictObjPutKeyList, Tcl_DictObjRemoveKeyList - manipulate Tcl objects as dictionaries

SYNOPSIS

#include <tcl.h>
Tcl_Obj *
Tcl_NewDictObj()
int
Tcl_DictObjGet(interp, dictPtr, keyPtr, valuePtrPtr)
int
Tcl_DictObjPut(interp, dictPtr, keyPtr, valuePtr)
int
Tcl_DictObjRemove(interp, dictPtr, keyPtr)
int
Tcl_DictObjSize(interp, dictPtr, sizePtr)
int
Tcl_DictObjFirst(interp, dictPtr, searchPtr, keyPtrPtr, valuePtrPtr, donePtr)
void
Tcl_DictObjNext(searchPtr, keyPtrPtr, valuePtrPtr, donePtr)
void
Tcl_DictObjDone(searchPtr)
int
Tcl_DictObjPutKeyList(interp, dictPtr, keyc, keyv, valuePtr)
int
Tcl_DictObjRemoveKeyList(interp, dictPtr, keyc, keyv)
<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interp *interp</code> (in)</td>
<td>If an error occurs while converting an object to be a dictionary object, an error message is left in the interpreter's result object unless <code>interp</code> is NULL.</td>
</tr>
<tr>
<td><code>Tcl_Obj *dictPtr</code> (in/out)</td>
<td>Points to the dictionary object to be manipulated. If <code>dictPtr</code> does not already point to a dictionary object, an attempt will be made to convert it to one.</td>
</tr>
<tr>
<td><code>Tcl_Obj *keyPtr</code> (in)</td>
<td>Points to the key for the key/value pair being manipulated within the dictionary object.</td>
</tr>
<tr>
<td><code>Tcl_Obj **keyPtrPtr</code> (out)</td>
<td>Points to a variable that will have the key from a key/value pair placed within it. May be NULL to indicate that the caller is not interested in the key.</td>
</tr>
<tr>
<td><code>Tcl_Obj *valuePtr</code> (in)</td>
<td>Points to the value for the key/value pair being manipulate within the dictionary object (or sub-object, in the case of <code>Tcl_DictObjPutKeyList</code>).</td>
</tr>
</tbody>
</table>
| `Tcl_Obj **valuePtrPtr` (out) | Points to a variable that will have the value from a key/value pair placed within it. For }
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *sizePtr (out)</td>
<td>Points to a variable that will have the number of key/value pairs contained within the dictionary placed within it.</td>
</tr>
<tr>
<td>Tcl_DictSearch *searchPtr (in/out)</td>
<td>Pointer to record to use to keep track of progress in enumerating all key/value pairs in a dictionary. The contents of the record will be initialized by the call to <strong>Tcl_DictObjFirst</strong>. If the enumerating is to be terminated before all values in the dictionary have been returned, the search record must be passed to <strong>Tcl_DictObjDone</strong> to enable the internal locks to be released.</td>
</tr>
<tr>
<td>int *donePtr (out)</td>
<td>Points to a variable that will have a non-zero value written into it when the enumeration of the key/value pairs in a dictionary has completed, and a zero otherwise.</td>
</tr>
</tbody>
</table>
int keyc (in)

Indicates the number of keys that will be supplied in the keyv array.

Tcl_Obj *const *keyv (in)

Array of keyc pointers to objects that
Tcl_DictObjPutKeyList and
Tcl_DictObjRemoveKeyList will use to locate the key/value pair to
manipulate within the sub-dictionaries of the main
dictionary object passed to them.

DESCRIPTION

Tcl dictionary objects have an internal representation that supports efficient mapping from keys to values and which guarantees that the particular ordering of keys within the dictionary remains the same modulo any keys being deleted (which removes them from the order) or added (which adds them to the end of the order). If reinterpreted as a list, the values at the even-valued indices in the list will be the keys of the dictionary, and each will be followed (in the odd-valued index) by the value associated with that key.

The procedures described in this man page are used to create, modify, index, and iterate over dictionary objects from C code.

Tcl_NewDictObj creates a new, empty dictionary object. The string representation of the object will be invalid, and the reference count of the object will be zero.

Tcl_DictObjGet looks up the given key within the given dictionary and writes a pointer to the value associated with that key into the variable pointed to by valuePtrPtr, or a NULL if the key has no mapping within
the dictionary. The result of this procedure is **TCL_OK**, or **TCL_ERROR** if the `dictPtr` cannot be converted to a dictionary.

**Tcl_DictObjPut** updates the given dictionary so that the given key maps to the given value; any key may exist at most once in any particular dictionary. The dictionary must not be shared, but the key and value may be. This procedure may increase the reference count of both key and value if it proves necessary to store them. Neither key nor value should be NULL. The result of this procedure is **TCL_OK**, or **TCL_ERROR** if the `dictPtr` cannot be converted to a dictionary.

**Tcl_DictObjRemove** updates the given dictionary so that the given key has no mapping to any value. The dictionary must not be shared, but the key may be. The key actually stored in the dictionary will have its reference count decremented if it was present. It is not an error if the key did not previously exist. The result of this procedure is **TCL_OK**, or **TCL_ERROR** if the `dictPtr` cannot be converted to a dictionary.

**Tcl_DictObjSize** updates the given variable with the number of key/value pairs currently in the given dictionary. The result of this procedure is **TCL_OK**, or **TCL_ERROR** if the `dictPtr` cannot be converted to a dictionary.

**Tcl_DictObjFirst** commences an iteration across all the key/value pairs in the given dictionary, placing the key and value in the variables pointed to by the `keyPtrPtr` and `valuePtrPtr` arguments (which may be NULL to indicate that the caller is uninterested in they key or variable respectively.) The next key/value pair in the dictionary may be retrieved with **Tcl_DictObjNext**. Concurrent updates of the dictionary's internal representation will not modify the iteration processing unless the dictionary is unshared, when this will trigger premature termination of the iteration instead (which Tcl scripts cannot trigger via the `dict` command.) The `searchPtr` argument points to a piece of context that is used to identify which particular iteration is being performed, and is initialized by the call to **Tcl_DictObjFirst**. The `donePtr` argument points to a variable that is updated to be zero if there are further key/value pairs to be iterated over, or non-zero if the iteration is complete. The order of iteration is implementation-defined. If the `dictPtr` argument
cannot be converted to a dictionary, \texttt{Tcl DictObjFirst} returns \texttt{TCL\_ERROR} and the iteration is not commenced, and otherwise it returns \texttt{TCL\_OK}.

When \texttt{Tcl DictObjFirst} is called upon a dictionary, a lock is placed on the dictionary to enable that dictionary to be iterated over safely without regard for whether the dictionary is modified during the iteration. Because of this, once the iteration over a dictionary's keys has finished (whether because all values have been iterated over as indicated by the variable indicated by the \texttt{donePtr} argument being set to one, or because no further values are required) the \texttt{Tcl DictObjDone} function must be called with the same \texttt{searchPtr} as was passed to \texttt{Tcl DictObjFirst} so that the internal locks can be released. Once a particular \texttt{searchPtr} is passed to \texttt{Tcl DictObjDone}, passing it to \texttt{Tcl DictObjNext} (without first initializing it with \texttt{Tcl DictObjFirst}) will result in no values being produced and the variable pointed to by \texttt{donePtr} being set to one. It is safe to call \texttt{Tcl DictObjDone} multiple times on the same \texttt{searchPtr} for each call to \texttt{Tcl DictObjFirst}.

The procedures \texttt{Tcl DictObjPutKeyList} and \texttt{Tcl DictObjRemoveKeyList} are the close analogues of \texttt{Tcl DictObjPut} and \texttt{Tcl DictObjRemove} respectively, except that instead of working with a single dictionary, they are designed to operate on a nested tree of dictionaries, with inner dictionaries stored as values inside outer dictionaries. The \texttt{keyc} and \texttt{keyv} arguments specify a list of keys (with outermost keys first) that acts as a path to the key/value pair to be affected. Note that there is no corresponding operation for reading a value for a path as this is easy to construct from repeated use of \texttt{Tcl DictObjGet}. With \texttt{Tcl DictObjPutKeyList}, nested dictionaries are created for non-terminal keys where they do not already exist. With \texttt{Tcl DictObjRemoveKeyList}, all non-terminal keys must exist and have dictionaries as their values.

\begin{example}

Using the dictionary iteration interface to search determine if there is a key that maps to itself:

\end{example}
Tcl_DictSearch search;
Tcl_Obj *key, *value;
int done;

/*
 * Assume interp and objPtr are parameters. This is
 * idiomatic way to start an iteration over the dict
 * sets a lock on the internal representation that e
 * there are no concurrent modification issues when
 * reference count management is also used. The loc
 * released automatically when the loop is finished, 
 * be released manually when an exceptional exit fro
 * is performed. However it is safe to try to releas
 * even if we've finished iterating over the loop. 
 */
if (Tcl_DictObjFirst(interp, objPtr, &search,
                &key, &value, &done) != TCL_OK) {
    return TCL_ERROR;
}
for (; !done ; Tcl_DictObjNext(&search, &key, &value
/*
   * Note that strcmp() is not a good way of compa
   * objects and is just used here for demonstrati
   * purposes.
   */
   if (!strcmp(Tcl_GetString(key), Tcl_GetString(va
      break;
  }
}
Tcl_DictObjDone(&search);
Tcl_SetObjResult(interp, Tcl_NewBooleanObj(!done));
return TCL_OK;
**Tcl NewObj, Tcl DecrRefCount, Tcl IncrRefCount, Tcl InitObjHashTable**

**KEYWORDS**

dict, dict object, dictionary, dictionary object, hash table, iteration, object

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2003 Donal K. Fellows
NAME
Tcl_StandardChannels - How the Tcl library deals with the standard channels

DESCRIPTION
APPLICATION PROGRAMMING INTERFACES
INITIALIZATION OF TCL STANDARD CHANNELS
1)
2)
 (a)
 (b)
3)
RE-INITIALIZATION OF TCL STANDARD CHANNELS
SHELL-SPECIFIC DETAILS
tclsh
wish
SEE ALSO
KEYWORDS

NAME
Tcl_StandardChannels - How the Tcl library deals with the standard channels

DESCRIPTION
This page explains the initialization and use of standard channels in the Tcl library.

The term *standard channels* comes out of the Unix world and refers to the three channels automatically opened by the OS for each new application. They are *stdin*, *stdout* and *stderr*. The first is the standard input an application can read from, the other two refer to writable
channels, one for regular output and the other for error messages.

Tcl generalizes this concept in a cross-platform way and exposes standard channels to the script level.

**APPLICATION PROGRAMMING INTERFACES**

The public API procedures dealing directly with standard channels are `Tcl_GetStdChannel` and `Tcl_SetStdChannel`. Additional public APIs to consider are `Tcl_RegisterChannel`, `Tcl_CreateChannel` and `Tcl_GetChannel`.

**INITIALIZATION OF TCL STANDARD CHANNELS**

Standard channels are initialized by the Tcl library in three cases: when explicitly requested, when implicitly required before returning channel information, or when implicitly required during registration of a new channel.

These cases differ in how they handle unavailable platform-specific standard channels. (A channel is not “available” if it could not be successfully opened; for example, in a Tcl application run as a Windows NT service.)

1) A single standard channel is initialized when it is explicitly specified in a call to `Tcl_SetStdChannel`. The states of the other standard channels are unaffected.

   Missing platform-specific standard channels do not matter here. This approach is not available at the script level.

2) All uninitialized standard channels are initialized to platform-specific default values:

   (a) when open channels are listed with `Tcl_GetChannelNames`
(or the file channels script command), or

(b)

when information about any standard channel is requested with a call to \texttt{Tcl\_GetStdChannel}, or with a call to \texttt{Tcl\_GetChannel} which specifies one of the standard names (stdin, stdout and stderr).

In case of missing platform-specific standard channels, the Tcl standard channels are considered as initialized and then immediately closed. This means that the first three Tcl channels then opened by the application are designated as the Tcl standard channels.

3) All uninitialized standard channels are initialized to platform-specific default values when a user-requested channel is registered with \texttt{Tcl\_RegisterChannel}.

In case of unavailable platform-specific standard channels the channel whose creation caused the initialization of the Tcl standard channels is made a normal channel. The next three Tcl channels opened by the application are designated as the Tcl standard channels. In other words, of the first four Tcl channels opened by the application the second to fourth are designated as the Tcl standard channels.

\textbf{RE-INITIALIZATION OF TCL STANDARD CHANNELS}

Once a Tcl standard channel is initialized through one of the methods above, closing this Tcl standard channel will cause the next call to \texttt{Tcl\_CreateChannel} to make the new channel the new standard channel, too. If more than one Tcl standard channel was closed \texttt{Tcl\_CreateChannel} will fill the empty slots in the order stdin, stdout and stderr.

\texttt{Tcl\_CreateChannel} will not try to reinitialize an empty slot if that slot was not initialized before. It is this behavior which enables an application to employ method 1 of initialization, i.e. to create and
designate their own Tcl standard channels.

**SHELL-SPECIFIC DETAILS**

**tclsh**

The Tcl shell (or rather the function **Tcl_Main**, which forms the core of the shell's implementation) uses method 2 to initialize the standard channels.

**wish**

The windowing shell (or rather the function **Tk_MainEx**, which forms the core of the shell's implementation) uses method 1 to initialize the standard channels (See **Tk_InitConsoleChannels**) on non-Unix platforms. On Unix platforms, **Tk_MainEx** implicitly uses method 2 to initialize the standard channels.

**SEE ALSO**

**Tcl_CreateChannel**, **Tcl_RegisterChannel**, **Tcl_GetChannel**, **Tcl_GetStdChannel**, **Tcl_SetStdChannel**, **Tk_InitConsoleChannels**, **tclsh**, **wish**, **Tcl_Main**, **Tk_MainEx**

**KEYWORDS**

standard_channels

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2001 by ActiveState Corporation
Tcl_GetVersion - get the version of the library at runtime

SYNOPSIS

#include <tcl.h>
Tcl_GetVersion(major, minor, patchLevel, type)

ARGUMENTS

int *major (out) Major version number of the Tcl library.

int *minor (out) Minor version number of the Tcl library.

int *patchLevel (out) The patch level of the Tcl library (or alpha or beta number).

Tcl_ReleaseType *type (out) The type of release, also indicates the type of patch level. Can be one of TCL_ALPHA_RELEASE, TCL_BETA_RELEASE, or TCL_FINAL_RELEASE.

DESCRIPTION

Tcl_GetVersion should be used to query the version number of the Tcl
library at runtime. This is useful when using a dynamically loaded Tcl library or when writing a stubs-aware extension. For instance, if you write an extension that is linked against the Tcl stubs library, it could be loaded into a program linked to an older version of Tcl than you expected. Use **Tcl_GetVersion** to verify that fact, and possibly to change the behavior of your extension.

**Tcl_GetVersion** accepts NULL for any of the arguments. For instance if you do not care about the *patchLevel* of the library, pass a NULL for the *patchLevel* argument.

**KEYWORDS**

*version, patchlevel, major, minor, alpha, beta, release*
**NAME**

Tcl_StaticPackage - make a statically linked package available via the 'load' command

**SYNOPSIS**

```c
#include <tcl.h>
Tcl_StaticPackage(interp, pkgName, initProc, safeInitProc)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interp *interp (in)</code></td>
<td>If not NULL, points to an interpreter into which the package has already been loaded (i.e., the caller has already invoked the appropriate initialization procedure). NULL means the package has not yet been incorporated into any interpreter.</td>
</tr>
<tr>
<td>const char *pkgName (in)</td>
<td>Name of the package; should be properly capitalized (first letter upper-case, all others lower-case).</td>
</tr>
<tr>
<td><code>Tcl_PackageInitProc *initProc (in)</code></td>
<td>Procedure to invoke to incorporate this package</td>
</tr>
</tbody>
</table>
Procedure to call to incorporate this package into a safe interpreter (one that will execute untrusted scripts). NULL means the package cannot be used in safe interpreters.

**DESCRIPTION**

This procedure may be invoked to announce that a package has been linked statically with a Tcl application and, optionally, that it has already been loaded into an interpreter. Once `Tcl_StaticPackage` has been invoked for a package, it may be loaded into interpreters using the `load` command. `Tcl_StaticPackage` is normally invoked only by the `Tcl_ApplInit` procedure for the application, not by packages for themselves (`Tcl_StaticPackage` should only be invoked for statically loaded packages, and code in the package itself should not need to know whether the package is dynamically or statically loaded).

When the `load` command is used later to load the package into an interpreter, one of `initProc` and `safelnitProc` will be invoked, depending on whether the target interpreter is safe or not. `initProc` and `safelnitProc` must both match the following prototype:

```c
typedef int Tcl_PackageInitProc(Tcl_Interp *interp);
```

The `interp` argument identifies the interpreter in which the package is to be loaded. The initialization procedure must return `TCL_OK` or `TCL_ERROR` to indicate whether or not it completed successfully; in the event of an error it should set the interpreter's result to point to an error message. The result or error from the initialization procedure will
be returned as the result of the `load` command that caused the initialization procedure to be invoked.

**KEYWORDS**

initialization procedure, package, static linking

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME
Tcl_StringMatch, Tcl_StringCaseMatch - test whether a string matches a pattern

SYNOPSIS
#include <tcl.h>
int
Tcl_StringMatch(str, pattern)
int
Tcl_StringCaseMatch(str, pattern, flags)

ARGUMENTS

DESCRIPTION

KEYWORDS

NAME
Tcl_StringMatch, Tcl_StringCaseMatch - test whether a string matches a pattern

SYNOPSIS
#include <tcl.h>
int
Tcl_StringMatch(str, pattern)
int
Tcl_StringCaseMatch(str, pattern, flags)

ARGUMENTS

const char *str (in) String to test.

const char *pattern (in) Pattern to match against string. May contain special
DESCRIPTION

This utility procedure determines whether a string matches a given pattern. If it does, then Tcl_StringMatch returns 1. Otherwise Tcl_StringMatch returns 0. The algorithm used for matching is the same algorithm used in the string match Tcl command and is similar to the algorithm used by the C-shell for file name matching; see the Tcl manual entry for details.

In Tcl_StringCaseMatch, the algorithm is the same, but you have the option to make the matching case-insensitive. If you choose this (by passing TCL_MATCH_NOCASE), then the string and pattern are essentially matched in the lower case.

KEYWORDS

match, pattern, string

Copyright © 1995-1997 Roger E. Critchlow Jr.  
Copyright © 1989-1993 The Regents of the University of California.  
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tcl_EvalObjEx, Tcl_EvalFile, Tcl_EvalObjv, Tcl_Eval,
Tcl_EvalEx, Tcl_GlobalEval, Tcl_GlobalEvalObj, Tcl_VarEval,
Tcl_VarEvalVA - execute Tcl scripts

SYNOPSIS
#include <tcl.h>
int
Tcl_EvalObjEx(interp, objPtr, flags)
int
Tcl_EvalFile(interp, fileName)
int
Tcl_EvalObjv(interp, objc, objv, flags)
int
Tcl_Eval(interp, script)
int
Tcl_EvalEx(interp, script, numBytes, flags)
int
Tcl_GlobalEval(interp, script)
int
Tcl_GlobalEvalObj(interp, objPtr)
int
Tcl_VarEval(interp, part, part, ... (char *) NULL)
int
Tcl_VarEvalVA(interp, argList)

ARGUMENTS
DESCRIPTION
FLAG BITS
TCL_EVAL_DIRECT
TCL_EVAL_GLOBAL
MISCELLANEOUS DETAILS
KEYWORDS
NAME

Tcl_EvalObjEx, Tcl_EvalFile, Tcl_EvalObjv, Tcl_Eval, Tcl_EvalEx, Tcl_GlobalEval, Tcl_GlobalEvalObj, Tcl_VarEval, Tcl_VarEvalVA - execute Tcl scripts

SYNOPSIS

#include <tcl.h>
int
Tcl_EvalObjEx(interp, objPtr, flags)
int
Tcl_EvalFile(interp, fileName)
int
Tcl_EvalObjv(interp, objc, objv, flags)
int
Tcl_Eval(interp, script)
int
Tcl_EvalEx(interp, script, numBytes, flags)
int
Tcl_GlobalEval(interp, script)
int
Tcl_GlobalEvalObj(interp, objPtr)
int
Tcl_VarEval(interp, part, part, ... (char *) NULL)
int
Tcl_VarEvalVA(interp, argList)

ARGUMENTS

Tcl_Interp *interp (in) Interpreter in which to execute the script. The interpreter's result is modified to hold the result or error message from the script.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_Obj *objPtr (in)</td>
<td>A Tcl object containing the script to execute.</td>
</tr>
</tbody>
</table>
| int flags (in)          | ORed combination of flag bits that specify additional options. 
|                         | **TCL_EVAL_GLOBAL** and **TCL_EVAL_DIRECT** are currently supported.        |
| const char *fileName (in)| Name of a file containing a Tcl script.                                     |
| int objc (in)           | The number of objects in the array pointed to by objPtr; this is also the   |
|                         | number of words in the command.                                             |
| Tcl_Obj **objv (in)     | Points to an array of pointers to objects; each object holds the value of a |
|                         | single word in the command to execute.                                      |
| int numBytes (in)       | The number of bytes in script, not including any null terminating character.|
|                         | If -1, then all characters up to the first null byte are used.              |
| const char *script (in) | Points to first byte of script to execute (null-terminated and UTF-8).      |
| char *part (in)         | String forming part of a Tcl 
va_list argList (in)  An argument list which must have been initialized using va_start, and cleared using va_end.

DESCRIPTION

The procedures described here are invoked to execute Tcl scripts in various forms. Tcl_EvalObjEx is the core procedure and is used by many of the others. It executes the commands in the script stored in objPtr until either an error occurs or the end of the script is reached. If this is the first time objPtr has been executed, its commands are compiled into bytecode instructions which are then executed. The bytecodes are saved in objPtr so that the compilation step can be skipped if the object is evaluated again in the future.

The return value from Tcl_EvalObjEx (and all the other procedures described here) is a Tcl completion code with one of the values TCL_OK, TCL_ERROR, TCL_RETURN, TCL_BREAK, or TCL_CONTINUE, or possibly some other integer value originating in an extension. In addition, a result value or error message is left in interp's result; it can be retrieved using Tcl_GetObjResult.

Tcl_EvalFile reads the file given by fileName and evaluates its contents as a Tcl script. It returns the same information as Tcl_EvalObjEx. If the file could not be read then a Tcl error is returned to describe why the file could not be read. The eofchar for files is “\32” (^Z) for all platforms. If you require a “^Z” in code for string comparison, you can use “\032” or “\u001a”, which will be safely substituted by the Tcl interpreter into “^Z”.

Tcl_EvalObjv executes a single pre-parsed command instead of a script. The objc and objv arguments contain the values of the words for the Tcl command, one word in each object in objv. Tcl_EvalObjv evaluates the command and returns a completion code and result just like Tcl_EvalObjEx. The caller of Tcl_EvalObjv has to manage the
reference count of the elements of objv, insuring that the objects are valid until Tcl_EvalObjv returns.

Tcl_Eval is similar to Tcl_EvalObjEx except that the script to be executed is supplied as a string instead of an object and no compilation occurs. The string should be a proper UTF-8 string as converted by Tcl_ExternalToUtfDString or Tcl_ExternalToUtf when it is known to possibly contain upper ASCII characters whose possible combinations might be a UTF-8 special code. The string is parsed and executed directly (using Tcl_EvalObjv) instead of compiling it and executing the bytecodes. In situations where it is known that the script will never be executed again, Tcl_Eval may be faster than Tcl_EvalObjEx. Tcl_Eval returns a completion code and result just like Tcl_EvalObjEx. Note: for backward compatibility with versions before Tcl 8.0, Tcl_Eval copies the object result in interp to interp->result (use is deprecated) where it can be accessed directly. This makes Tcl_Eval somewhat slower than Tcl_EvalEx, which does not do the copy.

Tcl_EvalEx is an extended version of Tcl_Eval that takes additional arguments numBytes and flags. For the efficiency reason given above, Tcl_EvalEx is generally preferred over Tcl_Eval.

Tcl_GlobalEval and Tcl_GlobalEvalObj are older procedures that are now deprecated. They are similar to Tcl_EvalEx and Tcl_EvalObjEx except that the script is evaluated in the global namespace and its variable context consists of global variables only (it ignores any Tcl procedures that are active). These functions are equivalent to using the TCL_EVAL_GLOBAL flag (see below).

Tcl_VarEval takes any number of string arguments of any length, concatenates them into a single string, then calls Tcl_Eval to execute that string as a Tcl command. It returns the result of the command and also modifies interp->result in the same way as Tcl_Eval. The last argument to Tcl_VarEval must be NULL to indicate the end of arguments. Tcl_VarEval is now deprecated.

Tcl_VarEvalVA is the same as Tcl_VarEval except that instead of taking a variable number of arguments it takes an argument list. Like
Tcl_VarEval, Tcl_VarEvalVA is deprecated.

## FLAG BITS

Any ORed combination of the following values may be used for the flags argument to procedures such as Tcl_EvalObjEx:

### TCL_EVAL_DIRECT

This flag is only used by Tcl_EvalObjEx; it is ignored by other procedures. If this flag bit is set, the script is not compiled to bytecodes; instead it is executed directly as is done by Tcl_EvalEx. The TCL_EVAL_DIRECT flag is useful in situations where the contents of an object are going to change immediately, so the bytecodes will not be reused in a future execution. In this case, it is faster to execute the script directly.

### TCL_EVAL_GLOBAL

If this flag is set, the script is processed at global level. This means that it is evaluated in the global namespace and its variable context consists of global variables only (it ignores any Tcl procedures at are active).

## MISCELLANEOUS DETAILS

During the processing of a Tcl command it is legal to make nested calls to evaluate other commands (this is how procedures and some control structures are implemented). If a code other than TCL_OK is returned from a nested Tcl_EvalObjEx invocation, then the caller should normally return immediately, passing that same return code back to its caller, and so on until the top-level application is reached. A few commands, like `for`, will check for certain return codes, like TCL_BREAK and TCL_CONTINUE, and process them specially without returning.

Tcl_EvalObjEx keeps track of how many nested Tcl_EvalObjEx invocations are in progress for interp. If a code of TCL_RETURN, TCL_BREAK, or TCL_CONTINUE is about to be returned from the topmost Tcl_EvalObjEx invocation for interp, it converts the return code
to **TCL_ERROR** and sets *interp*'s result to an error message indicating that the **return**, **break**, or **continue** command was invoked in an inappropriate place. This means that top-level applications should never see a return code from **Tcl_EvalObjEx** other then **TCL_OK** or **TCL_ERROR**.

**KEYWORDS**

execute, file, global, object, result, script
**NAME**

Tcl_SubstObj - perform substitutions on Tcl objects

**SYNOPSIS**

```c
#include <tcl.h>
Tcl_Obj *
Tcl_SubstObj(interp, objPtr, flags)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interp *interp (in)</code></td>
<td>Interpreter in which to execute Tcl scripts and lookup variables. If an error occurs, the interpreter's result is modified to hold an error message.</td>
</tr>
<tr>
<td><code>Tcl_Obj *objPtr (in)</code></td>
<td>A Tcl object containing the string to perform substitutions on.</td>
</tr>
<tr>
<td><code>int flags (in)</code></td>
<td>ORed combination of flag bits that specify which substitutions to perform. The flags <code>TCL_SUBST_COMMANDS</code> <code>TCL_SUBST_VARIABLES</code> and</td>
</tr>
</tbody>
</table>
TCL_SUBST_BACKSLASHES are currently supported, and TCL_SUBST_ALL is provided as a convenience for the common case where all substitutions are desired.

**DESCRIPTION**

The Tcl_SubstObj function is used to perform substitutions on strings in the fashion of the subst command. It gets the value of the string contained in objPtr and scans it, copying characters and performing the chosen substitutions as it goes to an output object which is returned as the result of the function. In the event of an error occurring during the execution of a command or variable substitution, the function returns NULL and an error message is left in interp's result.

Three kinds of substitutions are supported. When the TCL_SUBST_BACKSLASHES bit is set in flags, sequences that look like backslash substitutions for Tcl commands are replaced by their corresponding character.

When the TCL_SUBST_VARIABLES bit is set in flags, sequences that look like variable substitutions for Tcl commands are replaced by the contents of the named variable.

When the TCL_SUBST_COMMANDS bit is set in flags, sequences that look like command substitutions for Tcl commands are replaced by the result of evaluating that script. Where an uncaught "continue exception" occurs during the evaluation of a command substitution, an empty string is substituted for the command. Where an uncaught "break exception" occurs during the evaluation of a command substitution, the result of the whole substitution on objPtr will be truncated at the point immediately before the start of the command substitution, and no characters will be added to the result or substitutions performed after that point.
SEE ALSO

subst

KEYWORDS

backslash substitution, command substitution, variable substitution

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2001 Donal K. Fellows
NAME

Tcl_Init - find and source initialization script

SYNOPSIS

```c
#include <tcl.h>
int Tcl_Init(interp)
```

ARGUMENTS

Tcl_Interp *interp (in) interpreter to initialize.

DESCRIPTION

Tcl_Init is a helper procedure that finds and sources the init.tcl script, which should exist somewhere on the Tcl library path.

Tcl_Init is typically called from Tcl_AppInit procedures.

SEE ALSO

Tcl_AppInit, Tcl_Main

KEYWORDS

application, initialization, interpreter

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
Tcl_TraceVar, Tcl_TraceVar2, Tcl_UntraceVar,
Tcl_UntraceVar2, Tcl_VarTraceInfo, Tcl_VarTraceInfo2 -
monitor accesses to a variable

SYNOPSIS
#include <tcl.h>
int
Tcl_TraceVar(interp, varName, flags, proc, clientData)
int
Tcl_TraceVar2(interp, name1, name2, flags, proc, clientData)
Tcl_UntraceVar(interp, varName, flags, proc, clientData)
Tcl_UntraceVar2(interp, name1, name2, flags, proc,
clientData)
ClientData
Tcl_VarTraceInfo(interp, varName, flags, proc,
prevClientData)
ClientData
Tcl_VarTraceInfo2(interp, name1, name2, flags, proc,
prevClientData)

ARGUMENTS

DESCRIPTION

TCL_GLOBAL_ONLY
TCL_NAMESPACE_ONLY
TCL_TRACE_READS
TCL_TRACE_WRITES
TCL_TRACE_UNSETS
TCL_TRACE_ARRAY
TCL_TRACE_RESULT_DYNAMIC
TCL_TRACE_RESULT_OBJECT

TWO-PART NAMES
ACCESSING VARIABLES DURING TRACES
CALLBACK TIMING
WHOLE-ARRAY TRACES
MULTIPLE TRACES
ERROR RETURNS
RESTRICTIONS
UNDEFINED VARIABLES
TCL_TRACE.Destroyed FLAG
TCL.INTERP.Destroyed
BUGS
KEYWORDS

NAME

Tcl_TraceVar, Tcl_TraceVar2, Tcl_UntraceVar, Tcl_UntraceVar2,
Tcl_VarTraceInfo, Tcl_VarTraceInfo2 - monitor accesses to a variable

SYNOPSIS

#include <tcl.h>
int
Tcl_TraceVar(interp, varName, flags, proc, clientData)
int
Tcl_TraceVar2(interp, name1, name2, flags, proc, clientData)
Tcl_UntraceVar(interp, varName, flags, proc, clientData)
Tcl_UntraceVar2(interp, name1, name2, flags, proc, clientData)
ClientData
Tcl_VarTraceInfo(interp, varName, flags, proc, prevClientData)
ClientData
Tcl_VarTraceInfo2(interp, name1, name2, flags, proc, prevClientData)

ARGUMENTS

Tcl_Interp *interp (in)
Interpreter containing variable.

const char *varName (in)
Name of variable. May refer to a scalar variable, to an array variable with no
int flags (in)  
OR-ed combination of the values  
TCL_TRACE_READS,  
TCL_TRACE_WRITES,  
TCL_TRACE_UNSETS,  
TCL_TRACE_ARRAY,  
TCL_GLOBAL_ONLY,  
TCL_NAMESPACE_ONLY,  
TCL_TRACE_RESULT_DYNAMIC  
and  
TCL_TRACE_RESULT_OE  
Not all flags are used by all procedures. See below for more information.

Tcl_VarTraceProc *proc (in)  
Procedure to invoke whenever one of the traced operations occurs.

ClientData clientData (in)  
Arbitrary one-word value to pass to proc.

cnst char *name1 (in)  
Name of scalar or array variable (without array index).

cnst char *name2 (in)  
For a trace on an element of an array, gives the index of the element. For traces on scalar variables or on whole arrays, is NULL.

ClientData prevClientData (in)  
If non-NULL, gives last
value returned by 
Tcl_VarTraceInfo or 
Tcl_VarTraceInfo2, so this 
call will return information 
about next trace. If NULL, 
this call will return 
information about first 
trace.

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
</tr>
</thead>
</table>

**Tcl_TraceVar** allows a C procedure to monitor and control access to a 
Tcl variable, so that the C procedure is invoked whenever the variable is 
read or written or unset. If the trace is created successfully then 
**Tcl_TraceVar** returns **TCL_OK**. If an error occurred (e.g. *varName* 
specifies an element of an array, but the actual variable is not an array) 
then **TCL_ERROR** is returned and an error message is left in the 
interpreter's result.

The *flags* argument to **Tcl_TraceVar** indicates when the trace 
procedure is to be invoked and provides information for setting up the 
trace. It consists of an OR-ed combination of any of the following 
values:

**TCL_GLOBAL_ONLY**

 Normally, the variable will be looked up at the current level of 
procedure call; if this bit is set then the variable will be looked up at 
global level, ignoring any active procedures.

**TCL_NAMESPACE_ONLY**

 Normally, the variable will be looked up at the current level of 
procedure call; if this bit is set then the variable will be looked up in 
the current namespace, ignoring any active procedures.

**TCL_TRACE_READS**

 Invoke *proc* whenever an attempt is made to read the variable.
TCL_TRACE_WRITES
Invoke proc whenever an attempt is made to modify the variable.

TCL_TRACE_UNSETS
Invoke proc whenever the variable is unset. A variable may be unset either explicitly by an unset command, or implicitly when a procedure returns (its local variables are automatically unset) or when the interpreter is deleted (all variables are automatically unset).

TCL_TRACE_ARRAY
Invoke proc whenever the array command is invoked. This gives the trace procedure a chance to update the array before array names or array get is called. Note that this is called before an array set, but that will trigger write traces.

TCL_TRACE_RESULT_DYNAMIC
The result of invoking the proc is a dynamically allocated string that will be released by the Tcl library via a call to ckfree. Must not be specified at the same time as TCL_TRACE_RESULT_OBJECT.

TCL_TRACE_RESULT_OBJECT
The result of invoking the proc is a Tcl_Obj* (cast to a char*) with a reference count of at least one. The ownership of that reference will be transferred to the Tcl core for release (when the core has finished with it) via a call to Tcl_DecrRefCount. Must not be specified at the same time as TCL_TRACE_RESULT_DYNAMIC.

Whenever one of the specified operations occurs on the variable, proc will be invoked. It should have arguments and result that match the type Tcl_VarTraceProc:

typedef char *Tcl_VarTraceProc(
    ClientData clientData,
    Tcl_Interp *interp,
    char *name1,
    char *name2,
The `clientData` and `interp` parameters will have the same values as those passed to `Tcl_TraceVar` when the trace was created. `ClientData` typically points to an application-specific data structure that describes what to do when `proc` is invoked. `Name1` and `name2` give the name of the traced variable in the normal two-part form (see the description of `Tcl_TraceVar2` below for details). `Flags` is an OR-ed combination of bits providing several pieces of information. One of the bits `TCL_TRACE_READS`, `TCL_TRACE_WRITES`, `TCL_TRACE_ARRAY`, or `TCL_TRACE_UNSETS` will be set in `flags` to indicate which operation is being performed on the variable. The bit `TCL_GLOBAL_ONLY` will be set whenever the variable being accessed is a global one not accessible from the current level of procedure call: the trace procedure will need to pass this flag back to variable-related procedures like `Tcl_GetVar` if it attempts to access the variable. The bit `TCL_NAMESPACE_ONLY` will be set whenever the variable being accessed is a namespace one not accessible from the current level of procedure call: the trace procedure will need to pass this flag back to variable-related procedures like `Tcl_GetVar` if it attempts to access the variable. The bit `TCL_TRACE_DESTROYED` will be set in `flags` if the trace is about to be destroyed; this information may be useful to `proc` so that it can clean up its own internal data structures (see the section `TCL_TRACE_DESTROYED` below for more details). Lastly, the bit `TCL_INTERP_DESTROYED` will be set if the entire interpreter is being destroyed. When this bit is set, `proc` must be especially careful in the things it does (see the section `TCL_INTERP_DESTROYED` below). The trace procedure's return value should normally be NULL; see `ERROR_RETURNS` below for information on other possibilities.

`Tcl_UntraceVar` may be used to remove a trace. If the variable specified by `interp`, `varName`, and `flags` has a trace set with `flags`, `proc`, and `clientData`, then the corresponding trace is removed. If no such trace exists, then the call to `Tcl_UntraceVar` has no effect. The same bits are valid for `flags` as for calls to `Tcl_TraceVar`.

`Tcl_VarTraceInfo` may be used to retrieve information about traces set
on a given variable. The return value from `Tcl_VarTraceInfo` is the `clientData` associated with a particular trace. The trace must be on the variable specified by the `interp`, `varName`, and `flags` arguments (only the `TCL_GLOBAL_ONLY` and `TCL_NAMESPACE_ONLY` bits from `flags` is used; other bits are ignored) and its trace procedure must the same as the `proc` argument. If the `prevClientData` argument is NULL then the return value corresponds to the first (most recently created) matching trace, or NULL if there are no matching traces. If the `prevClientData` argument is not NULL, then it should be the return value from a previous call to `Tcl_VarTraceInfo`. In this case, the new return value will correspond to the next matching trace after the one whose `clientData` matches `prevClientData`, or NULL if no trace matches `prevClientData` or if there are no more matching traces after it. This mechanism makes it possible to step through all of the traces for a given variable that have the same `proc`.

**TWO-PART NAMES**

The procedures `Tcl_TraceVar2`, `Tcl_UntraceVar2`, and `Tcl_VarTraceInfo2` are identical to `Tcl_TraceVar`, `Tcl_UntraceVar`, and `Tcl_VarTraceInfo`, respectively, except that the name of the variable consists of two parts. `Name1` gives the name of a scalar variable or array, and `name2` gives the name of an element within an array. When `name2` is NULL, `name1` may contain both an array and an element name: if the name contains an open parenthesis and ends with a close parenthesis, then the value between the parentheses is treated as an element name (which can have any string value) and the characters before the first open parenthesis are treated as the name of an array variable. If `name2` is NULL and `name1` does not refer to an array element it means that either the variable is a scalar or the trace is to be set on the entire array rather than an individual element (see WHOLE-ARRAY TRACES below for more information).

**ACCESSING VARIABLES DURING TRACES**

During read, write, and array traces, the trace procedure can read, write, or unset the traced variable using `Tcl_GetVar2`, `Tcl_SetVar2`, and other procedures. While `proc` is executing, traces are temporarily
disabled for the variable, so that calls to *Tcl_GetVar2* and *Tcl_SetVar2* will not cause *proc* or other trace procedures to be invoked again. Disabling only occurs for the variable whose trace procedure is active; accesses to other variables will still be traced. However, if a variable is unset during a read or write trace then unset traces will be invoked.

During unset traces the variable has already been completely expunged. It is possible for the trace procedure to read or write the variable, but this will be a new version of the variable. Traces are not disabled during unset traces as they are for read and write traces, but existing traces have been removed from the variable before any trace procedures are invoked. If new traces are set by unset trace procedures, these traces will be invoked on accesses to the variable by the trace procedures.

**CALLBACK TIMING**

When read tracing has been specified for a variable, the trace procedure will be invoked whenever the variable's value is read. This includes *set* Tcl commands, $-notation in Tcl commands, and invocations of the *Tcl_GetVar* and *Tcl_GetVar2* procedures. *Proc* is invoked just before the variable's value is returned. It may modify the value of the variable to affect what is returned by the traced access. If it unsets the variable then the access will return an error just as if the variable never existed.

When write tracing has been specified for a variable, the trace procedure will be invoked whenever the variable's value is modified. This includes *set* commands, commands that modify variables as side effects (such as *catch* and *scan*), and calls to the *Tcl_SetVar* and *Tcl_SetVar2* procedures. *Proc* will be invoked after the variable's value has been modified, but before the new value of the variable has been returned. It may modify the value of the variable to override the change and to determine the value actually returned by the traced access. If it deletes the variable then the traced access will return an empty string.

When array tracing has been specified, the trace procedure will be invoked at the beginning of the array command implementation, before
any of the operations like get, set, or names have been invoked. The trace procedure can modify the array elements with \texttt{Tcl\_SetVar} and \texttt{Tcl\_SetVar2}.

When unset tracing has been specified, the trace procedure will be invoked whenever the variable is destroyed. The traces will be called after the variable has been completely unset.

### WHOLE-ARRAY TRACES

If a call to \texttt{Tcl\_TraceVar} or \texttt{Tcl\_TraceVar2} specifies the name of an array variable without an index into the array, then the trace will be set on the array as a whole. This means that \texttt{proc} will be invoked whenever any element of the array is accessed in the ways specified by \texttt{flags}. When an array is unset, a whole-array trace will be invoked just once, with \texttt{name1} equal to the name of the array and \texttt{name2} NULL; it will not be invoked once for each element.

### MULTIPLE TRACES

It is possible for multiple traces to exist on the same variable. When this happens, all of the trace procedures will be invoked on each access, in order from most-recently-created to least-recently-created. When there exist whole-array traces for an array as well as traces on individual elements, the whole-array traces are invoked before the individual-element traces. If a read or write trace unsets the variable then all of the unset traces will be invoked but the remainder of the read and write traces will be skipped.

### ERROR RETURNS

Under normal conditions trace procedures should return NULL, indicating successful completion. If \texttt{proc} returns a non-NULL value it signifies that an error occurred. The return value must be a pointer to a static character string containing an error message, unless (exactly one of) the \texttt{TCL\_TRACE\_RESULT\_DYNAMIC} and \texttt{TCL\_TRACE\_RESULT\_OBJECT} flags is set, which specify that the result is either a dynamic string (to be released with \texttt{ckfree}) or a
Tcl_Obj* (cast to char* and to be released with Tcl_DecrRefCount) containing the error message. If a trace procedure returns an error, no further traces are invoked for the access and the traced access aborts with the given message. Trace procedures can use this facility to make variables read-only, for example (but note that the value of the variable will already have been modified before the trace procedure is called, so the trace procedure will have to restore the correct value).

The return value from proc is only used during read and write tracing. During unset traces, the return value is ignored and all relevant trace procedures will always be invoked.

**RESTRICTIONS**

A trace procedure can be called at any time, even when there is a partially formed result in the interpreter's result area. If the trace procedure does anything that could damage this result (such as calling Tcl_Eval) then it must save the original values of the interpreter's result and freeProc fields and restore them before it returns.

**UNDEFINED VARIABLES**

It is legal to set a trace on an undefined variable. The variable will still appear to be undefined until the first time its value is set. If an undefined variable is traced and then unset, the unset will fail with an error (“no such variable”), but the trace procedure will still be invoked.

**TCL_TRACE_DESTROYED FLAG**

In an unset callback to proc, the TCL_TRACE_DESTROYED bit is set in flags if the trace is being removed as part of the deletion. Traces on a variable are always removed whenever the variable is deleted; the only time TCL_TRACE_DESTROYED is not set is for a whole-array trace invoked when only a single element of an array is unset.

**TCL_INTERP_DESTROYED**

When an interpreter is destroyed, unset traces are called for all of its
variables. The **TCL_INTERP_DESTROYED** bit will be set in the *flags* argument passed to the trace procedures. Trace procedures must be extremely careful in what they do if the **TCL_INTERP_DESTROYED** bit is set. It is not safe for the procedures to invoke any Tcl procedures on the interpreter, since its state is partially deleted. All that trace procedures should do under these circumstances is to clean up and free their own internal data structures.

**BUGS**

Tcl does not do any error checking to prevent trace procedures from misusing the interpreter during traces with **TCL_INTERP_DESTROYED** set.

Array traces are not yet integrated with the Tcl **info exists** command, nor is there Tcl-level access to array traces.

**KEYWORDS**

clientData, trace, variable
NAME

Tcl_DoOneEvent - wait for events and invoke event handlers

SYNOPSIS

#include <tcl.h>

int

Tcl_DoOneEvent(flags)

ARGUMENTS

DESCRIPTION

TCL_WINDOW_EVENTS -
TCL_FILE_EVENTS -
TCL_TIMER_EVENTS -
TCL_IDLE_EVENTS -
TCL_ALL_EVENTS -
TCL_DONT_WAIT -

KEYWORDS

NAME

Tcl_DoOneEvent - wait for events and invoke event handlers

SYNOPSIS

#include <tcl.h>

int

Tcl_DoOneEvent(flags)

ARGUMENTS

int flags (in)

This parameter is normally zero. It may be an OR-ed combination of any of the following flag bits:
DESCRIPTION

This procedure is the entry point to Tcl's event loop; it is responsible for waiting for events and dispatching event handlers created with procedures such as Tk_CreateEventHandler, Tcl_CreateFileHandler, Tcl_CreateTimerHandler, and Tcl_DoWhenIdle. Tcl_DoOneEvent checks to see if events are already present on the Tcl event queue; if so, it calls the handler(s) for the first (oldest) event, removes it from the queue, and returns. If there are no events ready to be handled, then Tcl_DoOneEvent checks for new events from all possible sources. If any are found, it puts all of them on Tcl's event queue, calls handlers for the first event on the queue, and returns. If no events are found, Tcl_DoOneEvent checks for Tcl_DoWhenIdle callbacks; if any are found, it invokes all of them and returns. Finally, if no events or idle callbacks have been found, then Tcl_DoOneEvent sleeps until an event occurs; then it adds any new events to the Tcl event queue, calls handlers for the first event, and returns. The normal return value is 1 to signify that some event was processed (see below for other alternatives).

If the flags argument to Tcl_DoOneEvent is non-zero, it restricts the kinds of events that will be processed by Tcl_DoOneEvent. Flags may be an OR-ed combination of any of the following bits:

TCL_WINDOW_EVENTS -
    Process window system events.

TCL_FILE_EVENTS -
    Process file events.
**TCL_TIMER_EVENTS** -  
Process timer events.

**TCL_IDLE_EVENTS** -  
Process idle callbacks.

**TCL_ALL_EVENTS** -  
Process all kinds of events: equivalent to OR-ing together all of the above flags or specifying none of them.

**TCL_DONT_WAIT** -  
Do not sleep: process only events that are ready at the time of the call.

If any of the flags **TCL_WINDOW_EVENTS**, **TCL_FILE_EVENTS**, **TCL_TIMER_EVENTS**, or **TCL_IDLE_EVENTS** is set, then the only events that will be considered are those for which flags are set. Setting none of these flags is equivalent to the value **TCL_ALL_EVENTS**, which causes all event types to be processed. If an application has defined additional event sources with **Tcl_CreateEventSource**, then additional flag values may also be valid, depending on those event sources.

The **TCL_DONT_WAIT** flag causes **Tcl_DoOneEvent** not to put the process to sleep: it will check for events but if none are found then it returns immediately with a return value of 0 to indicate that no work was done. **Tcl_DoOneEvent** will also return 0 without doing anything if the only alternative is to block forever (this can happen, for example, if **flags** is **TCL_IDLE_EVENTS** and there are no **Tcl_DoWhenIdle** callbacks pending, or if no event handlers or timer handlers exist).

**Tcl_DoOneEvent** may be invoked recursively. For example, it is possible to invoke **Tcl_DoOneEvent** recursively from a handler called by **Tcl_DoOneEvent**. This sort of operation is useful in some modal situations, such as when a notification dialog has been popped up and an application wishes to wait for the user to click a button in the dialog before doing anything else.
KEYWORDS

callback, event, handler, idle, timer
NAME
Tcl_DumpActiveMemory, Tcl_InitMemory, Tcl_ValidateAllMemory - Validated memory allocation interface

SYNOPSIS
#include <tcl.h>
int
Tcl_DumpActiveMemory(fileName)
void
Tcl_InitMemory(interp)
void
Tcl_ValidateAllMemory(fileName, line)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS
ARGUMENTS

**Tcl_Interp** *interp* (in) Tcl interpreter in which to add commands.

const char *fileName (in) For **Tcl_DumpActiveMemory**, name of the file to which memory information will be written. For **Tcl.ValidateAllMemory**, name of the file from which the call is being made (normally __FILE__).

int line (in) Line number at which the call to **Tcl.ValidateAllMemory** is made (normally __LINE__).

DESCRIPTION

These functions provide access to Tcl memory debugging information. They are only functional when Tcl has been compiled with **TCL_MEM_DEBUG** defined at compile-time. When **TCL_MEM_DEBUG** is not defined, these functions are all no-ops.

**Tcl.DumpActiveMemory** will output a list of all currently allocated memory to the specified file. The information output for each allocated block of memory is: starting and ending addresses (excluding guard zone), size, source file where **ckalloc** was called to allocate the block and line number in that file. It is especially useful to call **Tcl.DumpActiveMemory** after the Tcl interpreter has been deleted.
**Tcl_InitMemory** adds the Tcl **memory** command to the interpreter given by *interp*. **Tcl_InitMemory** is called by **Tcl_Main**.

**Tcl_ValidateAllMemory** forces a validation of the guard zones of all currently allocated blocks of memory. Normally validation of a block occurs when its freed, unless full validation is enabled, in which case validation of all blocks occurs when **ckalloc** and **ckfree** are called. This function forces the validation to occur at any point.

---

**SEE ALSO**

TCL_MEM_DEBUG, **memory**

**KEYWORDS**

memory, debug

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1992-1999 Karl Lehenbauer and Mark Diekhans.
Copyright © 2000 by Scriptics Corporation.
NAME

Tcl_TranslateFileName - convert file name to native form and replace tilde with home directory

SYNOPSIS

#include <tcl.h>
char *
Tcl_TranslateFileName(interp, name, bufferPtr)

ARGUMENTS

Tcl_Interp *interp (in)  Interpreter in which to report an error, if any.

const char *name (in)  File name, which may start with a “~”.

Tcl_DString *bufferPtr (in/out)  If needed, this dynamic string is used to store the new file name. At the time of the call it should be uninitialized or free. The caller must eventually call Tcl_DStringFree to free up anything stored here.

DESCRIPTION

This utility procedure translates a file name to a platform-specific form
which, after being converted to the appropriate encoding, is suitable for passing to the local operating system. In particular, it converts network names into native form and does tilde substitution.

However, with the advent of the newer **Tcl_FSGetNormalizedPath** and **Tcl_GetNativePath**, there is no longer any need to use this procedure. In particular, **Tcl_GetNativePath** performs all the necessary translation and encoding conversion, is virtual-filesystem aware, and caches the native result for faster repeated calls. Finally **Tcl_GetNativePath** does not require you to free anything afterwards.

If **Tcl_TranslateFileName** has to do tilde substitution or translate the name then it uses the dynamic string at *bufferPtr to hold the new string it generates. After **Tcl_TranslateFileName** returns a non-NULL result, the caller must eventually invoke **Tcl_DStringFree** to free any information placed in *bufferPtr. The caller need not know whether or not **Tcl_TranslateFileName** actually used the string; **Tcl_TranslateFileName** initializes *bufferPtr even if it does not use it, so the call to **Tcl_DStringFree** will be safe in either case.

If an error occurs (e.g. because there was no user by the given name) then NULL is returned and an error message will be left in the interpreter's result. When an error occurs, **Tcl_TranslateFileName** frees the dynamic string itself so that the caller need not call **Tcl_DStringFree**.

The caller is responsible for making sure that the interpreter's result has its default empty value when **Tcl_TranslateFileName** is invoked.

**SEE ALSO**

filename

**KEYWORDS**

file name, home directory, tilde, translate, user

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
Tcl_DStringInit, Tcl_DStringAppend,
Tcl_DStringAppendElement, Tcl_DStringStartSublist,
Tcl_DStringEndSublist, Tcl_DStringLength, Tcl_DStringValue,
Tcl_DStringSetLength, Tcl_DStringTrunc, Tcl_DStringFree,
Tcl_DStringResult, Tcl_DStringGetResult - manipulate dynamic strings
SYNOPSIS
#include <tcl.h>
Tcl_DStringInit(dsPtr)
char *
Tcl_DStringAppend(dsPtr, bytes, length)
char *
Tcl_DStringAppendElement(dsPtr, element)
Tcl_DStringStartSublist(dsPtr)
Tcl_DStringEndSublist(dsPtr)
int
Tcl_DStringLength(dsPtr)
char *
Tcl_DStringValue(dsPtr)
Tcl_DStringSetLength(dsPtr, newLength)
Tcl_DStringTrunc(dsPtr, newLength)
Tcl_DStringFree(dsPtr)
Tcl_DStringResult(interp, dsPtr)
Tcl_DStringGetResult(interp, dsPtr)
ARGUMENTS
DESCRIPTION
KEYWORDS
Tcl_DStringInit, Tcl_DStringAppend, Tcl_DStringAppendElement, Tcl_DStringStartSublist, Tcl_DStringEndSublist, Tcl_DStringLength, Tcl_DStringValue, Tcl_DStringSetLength, Tcl_DStringTrunc, Tcl_DStringFree, Tcl_DStringResult, Tcl_DStringGetResult - manipulate dynamic strings

**SYNOPSIS**

```c
#include <tcl.h>

Tcl_DStringInit(dsPtr)
char *
Tcl_DStringAppend(dsPtr, bytes, length)
char *
Tcl_DStringAppendElement(dsPtr, element)
Tcl_DStringStartSublist(dsPtr)
Tcl_DStringEndSublist(dsPtr)
int
Tcl_DStringLength(dsPtr)
char *
Tcl_DStringValue(dsPtr)
Tcl_DStringSetLength(dsPtr, newLength)
Tcl_DStringTrunc(dsPtr, newLength)
Tcl_DStringFree(dsPtr)
Tcl_DStringResult(interp, dsPtr)
Tcl_DStringGetResult(interp, dsPtr)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_DString *dsPtr (in/out)</td>
<td>Pointer to structure that is used to manage a dynamic string.</td>
</tr>
<tr>
<td>const char *bytes (in)</td>
<td>Pointer to characters to append to dynamic string.</td>
</tr>
<tr>
<td>const char *element (in)</td>
<td>Pointer to characters to append as list element to</td>
</tr>
</tbody>
</table>
dynamic string.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <strong>length</strong> (in)</td>
<td>Number of bytes from <em>bytes</em> to add to dynamic string. If -1, add all characters up to null terminating character.</td>
</tr>
<tr>
<td>int <strong>newLength</strong> (in)</td>
<td>New length for dynamic string, not including null terminating character.</td>
</tr>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp</em> (in/out)</td>
<td>Interpreter whose result is to be set from or moved to the dynamic string.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

Dynamic strings provide a mechanism for building up arbitrarily long strings by gradually appending information. If the dynamic string is short then there will be no memory allocation overhead; as the string gets larger, additional space will be allocated as needed.

**Tcl_DStringInit** initializes a dynamic string to zero length. The Tcl_DString structure must have been allocated by the caller. No assumptions are made about the current state of the structure; anything already in it is discarded. If the structure has been used previously, **Tcl_DStringFree** should be called first to free up any memory allocated for the old string.

**Tcl_DStringAppend** adds new information to a dynamic string, allocating more memory for the string if needed. If *length* is less than zero then everything in *bytes* is appended to the dynamic string; otherwise *length* specifies the number of bytes to append. **Tcl_DStringAppend** returns a pointer to the characters of the new string. The string can also be retrieved from the *string* field of the Tcl_DString structure.
**Tcl_DStringAppendElement** is similar to **Tcl_DStringAppend** except that it does not take a *length* argument (it appends all of *element*) and it converts the string to a proper list element before appending.

**Tcl_DStringAppendElement** adds a separator space before the new list element unless the new list element is the first in a list or sub-list (i.e. either the current string is empty, or it contains the single character “{”, or the last two characters of the current string are “{”).

**Tcl_DStringAppendElement** returns a pointer to the characters of the new string.

**Tcl_DStringStartSublist** and **Tcl_DStringEndSublist** can be used to create nested lists. To append a list element that is itself a sublist, first call **Tcl_DStringStartSublist**, then call **Tcl_DStringAppendElement** for each of the elements in the sublist, then call **Tcl_DStringEndSublist** to end the sublist. **Tcl_DStringStartSublist** appends a space character if needed, followed by an open brace; **Tcl_DStringEndSublist** appends a close brace. Lists can be nested to any depth.

**Tcl_DStringLength** is a macro that returns the current length of a dynamic string (not including the terminating null character).

**Tcl_DStringValue** is a macro that returns a pointer to the current contents of a dynamic string.

**Tcl_DStringSetLength** changes the length of a dynamic string. If *newLength* is less than the string's current length, then the string is truncated. If *newLength* is greater than the string's current length, then the string will become longer and new space will be allocated for the string if needed. However, **Tcl_DStringSetLength** will not initialize the new space except to provide a terminating null character; it is up to the caller to fill in the new space. **Tcl_DStringSetLength** does not free up the string's storage space even if the string is truncated to zero length, so **Tcl_DStringFree** will still need to be called.

**Tcl_DStringTrunc** changes the length of a dynamic string. This procedure is now deprecated. **Tcl_DStringSetLength** should be used instead.
**Tcl_DStringFree** should be called when you are finished using the string. It frees up any memory that was allocated for the string and reinitializes the string's value to an empty string.

**Tcl_DStringResult** sets the result of *interp* to the value of the dynamic string given by *dsPtr*. It does this by moving a pointer from *dsPtr* to the interpreter's result. This saves the cost of allocating new memory and copying the string. **Tcl_DStringResult** also reinitializes the dynamic string to an empty string.

**Tcl_DStringGetResult** does the opposite of **Tcl_DStringResult**. It sets the value of *dsPtr* to the result of *interp* and it clears *interp*'s result. If possible it does this by moving a pointer rather than by copying the string.

**KEYWORDS**

append, dynamic string, free, result
NAME

Tcl_InitStubs - initialize the Tcl stubs mechanism

SYNOPSIS

#include <tcl.h>
const char *
Tcl_InitStubs(interp, version, exact)

ARGUMENTS

Tcl interpreter handle.

A version string consisting
INTRODUCTION

The Tcl stubs mechanism defines a way to dynamically bind extensions to a particular Tcl implementation at run time. This provides two significant benefits to Tcl users:

1) Extensions that use the stubs mechanism can be loaded into multiple versions of Tcl without being recompiled or relinked.

2) Extensions that use the stubs mechanism can be dynamically loaded into statically-linked Tcl applications.

The stubs mechanism accomplishes this by exporting function tables that define an interface to the Tcl API. The extension then accesses the Tcl API through offsets into the function table, so there are no direct references to any of the Tcl library's symbols. This redirection is transparent to the extension, so an extension writer can continue to use all public Tcl functions as documented.

The stubs mechanism requires no changes to applications incorporating Tcl interpreters. Only developers creating C-based Tcl extensions need
to take steps to use the stubs mechanism with their extensions.

Enabling the stubs mechanism for an extension requires the following steps:

1) Call `Tcl_InitStubs` in the extension before calling any other Tcl functions.

2) Define the `USE_TCL_STUBS` symbol. Typically, you would include the `-DUSE_TCL_STUBS` flag when compiling the extension.

3) Link the extension with the Tcl stubs library instead of the standard Tcl library. On Unix platforms, the library name is `libtclstub8.1.a`; on Windows platforms, the library name is `tclstub81.lib`.

If the extension also requires the Tk API, it must also call `Tk_InitStubs` to initialize the Tk stubs interface and link with the Tk stubs libraries. See the `Tk_InitStubs` page for more information.

**DESCRIPTION**

`Tcl_InitStubs` attempts to initialize the stub table pointers and ensure that the correct version of Tcl is loaded. In addition to an interpreter handle, it accepts as arguments a version number and a Boolean flag indicating whether the extension requires an exact version match or not. If `exact` is 0, then the extension is indicating that newer versions of Tcl are acceptable as long as they have the same major version number as `version`; non-zero means that only the specified `version` is acceptable. `Tcl_InitStubs` returns a string containing the actual version of Tcl satisfying the request, or NULL if the Tcl version is not acceptable, does not support stubs, or any other error condition occurred.

**SEE ALSO**

`Tk_InitStubs`
<table>
<thead>
<tr>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>stubs</td>
</tr>
</tbody>
</table>
NAME
Tcl_Interp - client-visible fields of interpreter structures

SYNOPSIS
#include <tcl.h>
typedef struct {
    char *result;
    Tcl_FreeProc *freeProc;
    int errorLine;
} Tcl_Interp;
typedef void Tcl_FreeProc(char *blockPtr);

DESCRIPTION

The Tcl_CreateInterp procedure returns a pointer to a Tcl_Interp structure. This pointer is then passed into other Tcl procedures to
process commands in the interpreter and perform other operations on the interpreter. Interpreter structures contain many fields that are used by Tcl, but only three that may be accessed by clients: `result`, `freeProc`, and `errorLine`.

**Note that access to all three fields, result, freeProc and errorLine is deprecated.** Use `Tcl_SetResult`, `Tcl_GetResult`, and `Tcl_GetReturnOptions` instead.

The `result` and `freeProc` fields are used to return results or error messages from commands. This information is returned by command procedures back to `Tcl_Eval`, and by `Tcl_Eval` back to its callers. The `result` field points to the string that represents the result or error message, and the `freeProc` field tells how to dispose of the storage for the string when it is not needed anymore. The easiest way for command procedures to manipulate these fields is to call procedures like `Tcl_SetResult` or `Tcl_AppendResult`; they will hide all the details of managing the fields. The description below is for those procedures that manipulate the fields directly.

Whenever a command procedure returns, it must ensure that the `result` field of its interpreter points to the string being returned by the command. The `result` field must always point to a valid string. If a command wishes to return no result then `interp->result` should point to an empty string. Normally, results are assumed to be statically allocated, which means that the contents will not change before the next time `Tcl_Eval` is called or some other command procedure is invoked. In this case, the `freeProc` field must be zero. Alternatively, a command procedure may dynamically allocate its return value (e.g. using `Tcl_Alloc`) and store a pointer to it in `interp->result`. In this case, the command procedure must also set `interp->freeProc` to the address of a procedure that can free the value, or `TCL_DYNAMIC` if the storage was allocated directly by Tcl or by a call to `Tcl_Alloc`. If `interp->freeProc` is non-zero, then Tcl will call `freeProc` to free the space pointed to by `interp->result` before it invokes the next command. If a client procedure overwrites `interp->result` when `interp->freeProc` is non-zero, then it is responsible for calling `freeProc` to free the old `interp->result` (the `Tcl_FreeResult` macro should be used for this purpose).
FreeProc should have arguments and result that match the Tcl_FreeProc declaration above: it receives a single argument which is a pointer to the result value to free. In most applications TCL_DYNAMIC is the only non-zero value ever used for freeProc. However, an application may store a different procedure address in freeProc in order to use an alternate memory allocator or in order to do other cleanup when the result memory is freed.

As part of processing each command, Tcl_Eval initializes interp->result and interp->freeProc just before calling the command procedure for the command. The freeProc field will be initialized to zero, and interp->result will point to an empty string. Commands that do not return any value can simply leave the fields alone. Furthermore, the empty string pointed to by result is actually part of an array of TCL_RESULT_SIZE characters (approximately 200). If a command wishes to return a short string, it can simply copy it to the area pointed to by interp->result. Or, it can use the sprintf procedure to generate a short result string at the location pointed to by interp->result.

It is a general convention in Tcl-based applications that the result of an interpreter is normally in the initialized state described in the previous paragraph. Procedures that manipulate an interpreter's result (e.g. by returning an error) will generally assume that the result has been initialized when the procedure is called. If such a procedure is to be called after the result has been changed, then Tcl_ResetResult should be called first to reset the result to its initialized state. The direct use of interp->result is strongly deprecated (see Tcl_SetResult).

The errorLine field is valid only after Tcl_Eval returns a TCL_ERROR return code. In this situation the errorLine field identifies the line number of the command being executed when the error occurred. The line numbers are relative to the command being executed: 1 means the first line of the command passed to Tcl_Eval, 2 means the second line, and so on. The errorLine field is typically used in conjunction with Tcl_AddErrorInfo to report information about where an error occurred. ErrorLine should not normally be modified except by Tcl_Eval.
KEYWORDS

free, initialized, interpreter, malloc, result
NAME
    Tcl_UniCharIsAlnum, Tcl_UniCharIsAlpha,
    Tcl_UniCharIsControl, Tcl_UniCharIsDigit,
    Tcl_UniCharIsGraph, Tcl_UniCharIsLower, Tcl_UniCharIsPrint,
    Tcl_UniCharIsPunct, Tcl_UniCharIsSpace,
    Tcl_UniCharIsUpper, Tcl_UniCharIsWordChar - routines for
    classification of Tcl_UniChar characters

SYNOPSIS
    #include <tcl.h>
    int
    Tcl_UniCharIsAlnum(ch)
    int
    Tcl_UniCharIsAlpha(ch)
    int
    Tcl_UniCharIsControl(ch)
    int
    Tcl_UniCharIsDigit(ch)
    int
    Tcl_UniCharIsGraph(ch)
    int
    Tcl_UniCharIsLower(ch)
    int
    Tcl_UniCharIsPrint(ch)
    int
    Tcl_UniCharIsPunct(ch)
    int
    Tcl_UniCharIsSpace(ch)
    int
    Tcl_UniCharIsUpper(ch)
    int
    Tcl_UniCharIsWordChar(ch)
ARGUMENTS
DESCRIPTION
CHARACTER CLASSES
KEYWORDS

NAME

Tcl_UniCharIsAlnum, Tcl_UniCharIsAlpha, Tcl_UniCharIsControl, Tcl_UniCharIsDigit, Tcl_UniCharIsGraph, Tcl_UniCharIsLower, Tcl_UniCharIsPrint, Tcl_UniCharIsPunct, Tcl_UniCharIsSpace, Tcl_UniCharIsUpper, Tcl_UniCharIsWordChar - routines for classification of Tcl_UniChar characters

SYNOPSIS

#include <tcl.h>
int
Tcl_UniCharIsAlnum(ch)
int
Tcl_UniCharIsAlpha(ch)
int
Tcl_UniCharIsControl(ch)
int
Tcl_UniCharIsDigit(ch)
int
Tcl_UniCharIsGraph(ch)
int
Tcl_UniCharIsLower(ch)
int
Tcl_UniCharIsPrint(ch)
int
Tcl_UniCharIsPunct(ch)
int
Tcl_UniCharIsSpace(ch)
int
Tcl_UniCharIsUpper(ch)
int
Tcl_UniCharIsWordChar(ch)
<table>
<thead>
<tr>
<th>ARGUMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>int ch (in)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>All of the routines described examine Tcl_UniChars and return a boolean value. A non-zero return value means that the character does belong to the character class associated with the called routine. The rest of this document just describes the character classes associated with the various routines.</td>
</tr>
</tbody>
</table>

Note: A Tcl_UniChar is a Unicode character represented as an unsigned, fixed-size quantity.

<table>
<thead>
<tr>
<th>CHARACTER CLASSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_UniCharIsAlnum tests if the character is an alphanumeric Unicode character.</td>
</tr>
<tr>
<td>Tcl_UniCharIsAlpha tests if the character is an alphabetic Unicode character.</td>
</tr>
<tr>
<td>Tcl_UniCharIsControl tests if the character is a Unicode control character.</td>
</tr>
<tr>
<td>Tcl_UniCharIsDigit tests if the character is a numeric Unicode character.</td>
</tr>
<tr>
<td>Tcl_UniCharIsGraph tests if the character is any Unicode print character except space.</td>
</tr>
<tr>
<td>Tcl_UniCharIsLower tests if the character is a lowercase Unicode character.</td>
</tr>
<tr>
<td>Tcl_UniCharIsPrint tests if the character is a Unicode print character.</td>
</tr>
</tbody>
</table>
**Tcl_UniCharIsPunct** tests if the character is a Unicode punctuation character.

**Tcl_UniCharIsSpace** tests if the character is a whitespace Unicode character.

**Tcl_UniCharIsUpper** tests if the character is an uppercase Unicode character.

**Tcl_UniCharIsWordChar** tests if the character is alphanumeric or a connector punctuation mark.

**KEYWORDS**

unicode, classification

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997 Sun Microsystems, Inc.
NAME
Tcl_LimitAddHandler, Tcl_LimitCheck, Tcl_LimitExceeded,
Tcl_LimitGetCommands, Tcl_LimitGetGranularity,
Tcl_LimitGetTime, Tcl_LimitReady, Tcl_LimitRemoveHandler,
Tcl_LimitSetCommands, Tcl_LimitSetGranularity,
Tcl_LimitSetTime, Tcl_LimitTypeEnabled,
Tcl_LimitTypeExceeded, Tcl_LimitTypeReset, Tcl_LimitTypeSet
- manage and check resource limits on interpreters

SYNOPSIS
#include <tcl.h>
int
Tcl_LimitCheck(interp)
int
Tcl_LimitReady(interp)
int
Tcl_LimitExceeded(interp)
int
Tcl_LimitTypeExceeded(interp, type)
int
Tcl_LimitTypeEnabled(interp, type)
void
Tcl_LimitTypeSet(interp, type)
void
Tcl_LimitTypeReset(interp, type)
int
Tcl_LimitGetCommands(interp)
void
Tcl_LimitSetCommands(interp, commandLimit)
void
Tcl_LimitGetTime(interp, timeLimitPtr)
void
Tcl_LimitSetTime(interp, timeLimitPtr)
    int
Tcl_LimitGetGranularity(interp, type)
    void
Tcl_LimitSetGranularity(interp, type, granularity)
    void
Tcl_LimitAddHandler(interp, type, handlerProc, clientData, deleteProc)
    void
Tcl_LimitRemoveHandler(interp, type, handlerProc, clientData)

ARGUMENTS
DESCRIPTION
LIMIT CHECKING API
LIMIT CONFIGURATION
LIMIT CALLBACKS
KEYWORDS

NAME

Tcl_LimitAddHandler, Tcl_LimitCheck, Tcl_LimitExceeded,
Tcl_LimitGetCommands, Tcl_LimitGetGranularity, Tcl_LimitGetTime,
Tcl_Limit Ready, Tcl_LimitRemoveHandler, Tcl_LimitSetCommands,
Tcl_LimitSetGranularity, Tcl_LimitSetTime, Tcl_LimitTypeEnabled,
Tcl_LimitTypeExceeded, Tcl_LimitTypeReset, Tcl_LimitTypeSet
- manage and check resource limits on interpreters

SYNOPSIS

#include <tcl.h>
    int
Tcl_LimitCheck(interp)
    int
Tcl_LimitReady(interp)
    int
Tcl_LimitExceeded(interp)
    int
Tcl_LimitTypeExceeded(interp, type)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_LimitTypeEnabled(interp, type)</code></td>
<td>Determines if the limit of type <code>type</code> is enabled for the interpreter <code>interp</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitTypeSet(interp, type)</code></td>
<td>Sets the limit of type <code>type</code> for the interpreter <code>interp</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitTypeReset(interp, type)</code></td>
<td>Resets the limit of type <code>type</code> for the interpreter <code>interp</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitGetCommands(interp)</code></td>
<td>Gets the commands limit for the interpreter <code>interp</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitSetCommands(interp, commandLimit)</code></td>
<td>Sets the commands limit for the interpreter <code>interp</code> to <code>commandLimit</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitGetTime(interp, timeLimitPtr)</code></td>
<td>Gets the time limit for the interpreter <code>interp</code> and stores it in the <code>timeLimitPtr</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitSetTime(interp, timeLimitPtr)</code></td>
<td>Sets the time limit for the interpreter <code>interp</code> to the value stored in the <code>timeLimitPtr</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitGetGranularity(interp, type)</code></td>
<td>Gets the granularity limit for the type <code>type</code> for the interpreter <code>interp</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitSetGranularity(interp, type, granularity)</code></td>
<td>Sets the granularity limit for the type <code>type</code> for the interpreter <code>interp</code> to the <code>granularity</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitAddHandler(interp, type, handlerProc, clientData, deleteProc)</code></td>
<td>Adds a handler for the type <code>type</code> for the interpreter <code>interp</code> with <code>handlerProc</code> and <code>clientData</code> and registers a <code>deleteProc</code>.</td>
</tr>
<tr>
<td><code>Tcl_LimitRemoveHandler(interp, type, handlerProc, clientData)</code></td>
<td>Removes a handler for the type <code>type</code> for the interpreter <code>interp</code> with <code>handlerProc</code> and <code>clientData</code>.</td>
</tr>
</tbody>
</table>

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interp *interp</code> (in)</td>
<td>Interpreter that the limit being managed applies to or that will have its limits checked.</td>
</tr>
<tr>
<td><code>int type</code> (in)</td>
<td>The type of limit that the operation refers to. This must be either <code>TCL_LIMIT_COMMANDS</code> or <code>TCL_LIMIT_TIME</code>.</td>
</tr>
</tbody>
</table>
**int commandLimit (in)**

The maximum number of commands (as reported by `info cmdcount`) that may be executed in the interpreter.

**Tcl_Time *timeLimitPtr (in/out)**

A pointer to a structure that will either have the new time limit read from (`Tcl_LimitSetTime`) or the current time limit written to (`Tcl_LimitGetTime`).

**int granularity (in)**

Divisor that indicates how often a particular limit should really be checked. Must be at least 1.

**Tcl_LimitHandlerProc *handlerProc (in)**

Function to call when a particular limit is exceeded. If the `handlerProc` removes or raises the limit during its processing, the limited interpreter will be permitted to continue to process after the handler returns. Many handlers may be attached to the same interpreter limit; their order of execution is not defined, and they must be identified by `handlerProc` and `clientData` when they are deleted.

**ClientData clientData (in)**

Arbitrary pointer-sized
Tcl's interpreter resource limit subsystem allows for close control over how much computation time a script may use, and is useful for cases where a program is divided into multiple pieces where some parts are more trusted than others (e.g. web application servers).

Every interpreter may have a limit on the wall-time for execution, and a limit on the number of commands that the interpreter may execute. Since checking of these limits is potentially expensive (especially the time limit), each limit also has a checking granularity, which is a divisor for an internal count of the number of points in the core where a check may be performed (which is immediately before executing a command and at an unspecified frequency between running commands, which can happen in empty-bodied \texttt{while} loops).

The final component of the limit engine is a callback scheme which allows for notifications of when a limit has been exceeded. These callbacks can just provide logging, or may allocate more resources to the interpreter to permit it to continue processing longer.

When a limit is exceeded (and the callbacks have run; the order of execution of the callbacks is unspecified) execution in the limited interpreter is stopped by raising an error and setting a flag that prevents the \texttt{catch} command in that interpreter from trapping that error. It is up to the context that started execution in that interpreter (typically a master interpreter) to handle the error.
To check the resource limits for an interpreter, call `Tcl_LimitCheck`, which returns `TCL_OK` if the limit was not exceeded (after processing callbacks) and `TCL_ERROR` if the limit was exceeded (in which case an error message is also placed in the interpreter result). That function should only be called when `Tcl_LimitReady` returns non-zero so that granularity policy is enforced. This API is designed to be similar in usage to `Tcl_AsyncReady` and `Tcl_AsyncInvoke`.

When writing code that may behave like `catch` in respect of errors, you should only trap an error if `Tcl_LimitExceeded` returns zero. If it returns non-zero, the interpreter is in a limit-exceeded state and errors should be allowed to propagate to the calling context. You can also check whether a particular type of limit has been exceeded using `Tcl_LimitTypeExceeded`.

To check whether a limit has been set (but not whether it has actually been exceeded) on an interpreter, call `Tcl_LimitTypeEnabled` with the type of limit you want to check. To enable a particular limit call `Tcl_LimitTypeSet`, and to disable a limit call `Tcl_LimitTypeReset`.

The level of a command limit may be set using `Tcl_LimitSetCommands`, and retrieved using `Tcl_LimitGetCommands`. Similarly for a time limit with `Tcl_LimitSetTime` and `Tcl_LimitGetTime` respectively, but with that API the time limit is copied from and to the Tcl_Time structure that the `timeLimitPtr` argument points to.

The checking granularity for a particular limit may be set using `Tcl_LimitSetGranularity` and retrieved using `Tcl_LimitGetGranularity`. Note that granularities must always be positive.

To add a handler callback to be invoked when a limit is exceeded, call
**Tcl_LimitAddHandler.** The *handlerProc* argument describes the function that will actually be called; it should have the following prototype:

```c
typedef void Tcl_LimitHandlerProc(
    ClientData clientData,
    Tcl_Interp  *interp);
```

The *clientData* argument to the handler will be whatever is passed to the *clientData* argument to *Tcl_LimitAddHandler*, and the *interp* is the interpreter that had its limit exceeded.

The *deleteProc* argument to *Tcl_LimitAddHandler* is a function to call to delete the *clientData* value. It may be *TCL_STATIC* or NULL if no deletion action is necessary, or *TCL_DYNAMIC* if all that is necessary is to free the structure with *Tcl_Free*. Otherwise, it should refer to a function with the following prototype:

```c
typedef void Tcl_LimitHandlerDeleteProc(
    ClientData clientData);
```

A limit handler may be deleted using *Tcl_LimitRemoveHandler*; the handler removed will be the first one found (out of the handlers added with *Tcl_LimitAddHandler*) with exactly matching *type*, *handlerProc* and *clientData* arguments. This function always invokes the *deleteProc* on the *clientData* (unless the *deleteProc* was NULL or *TCL_STATIC*).

**KEYWORDS**

`interpreter`, `resource`, `limit`, `commands`, `time`, `callback`
NAME
Tcl_UniCharToUpper, Tcl_UniCharToLower,
Tcl_UniCharToTitle, Tcl_UtfToUpper, Tcl_UtfToLower,
Tcl_UtfToTitle - routines for manipulating the case of Unicode
characters and UTF-8 strings
SYNOPSIS
#include <tcl.h>
Tcl_UniChar
Tcl_UniCharToUpper(ch)
Tcl_UniChar
Tcl_UniCharToLower(ch)
Tcl_UniChar
Tcl_UniCharToTitle(ch)
int
Tcl_UtfToUpper(str)
int
Tcl_UtfToLower(str)
int
Tcl_UtfToTitle(str)
ARGUMENTS
DESCRIPTION
BUGS
KEYWORDS
NAME
Tcl_UniCharToUpper, Tcl_UniCharToLower, Tcl_UniCharToTitle,
Tcl_UtfToUpper, Tcl_UtfToLower, Tcl_UtfToTitle - routines for
manipulating the case of Unicode characters and UTF-8 strings
SYNOPSIS
```
#include <tcl.h>
Tcl_UniChar Tcl_UniCharToUpper(ch)
Tcl_UniChar Tcl_UniCharToLower(ch)
Tcl_UniChar Tcl_UniCharToTitle(ch)
int Tcl_UtfToUpper(str)
int Tcl_UtfToLower(str)
int Tcl_UtfToTitle(str)

ARGUMENTS

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int ch</td>
<td>The Tcl_UniChar to be converted.</td>
</tr>
<tr>
<td>char *str</td>
<td>Pointer to UTF-8 string to be converted in place.</td>
</tr>
</tbody>
</table>

DESCRIPTION

The first three routines convert the case of individual Unicode characters:

If ch represents a lower-case character, Tcl_UniCharToUpper returns
the corresponding upper-case character. If no upper-case character is
defined, it returns the character unchanged.

If ch represents an upper-case character, Tcl_UniCharToLower returns
the corresponding lower-case character. If no lower-case character is
defined, it returns the character unchanged.

If ch represents a lower-case character, Tcl_UniCharToTitle returns the
corresponding title-case character. If no title-case character is defined,
it returns the corresponding upper-case character. If no upper-case character is defined, it returns the character unchanged. Title-case is defined for a small number of characters that have a different appearance when they are at the beginning of a capitalized word.

The next three routines convert the case of UTF-8 strings in place in memory:

**Tcl_UtfToUpper** changes every UTF-8 character in `str` to upper-case. Because changing the case of a character may change its size, the byte offset of each character in the resulting string may differ from its original location. **Tcl_UtfToUpper** writes a null byte at the end of the converted string. **Tcl_UtfToUpper** returns the new length of the string in bytes. This new length is guaranteed to be no longer than the original string length.

**Tcl_UtfToLower** is the same as **Tcl_UtfToUpper** except it turns each character in the string into its lower-case equivalent.

**Tcl_UtfToTitle** is the same as **Tcl_UtfToUpper** except it turns the first character in the string into its title-case equivalent and all following characters into their lower-case equivalents.

**BUGS**

At this time, the case conversions are only defined for the ISO8859-1 characters. Unicode characters above 0x00ff are not modified by these routines.

**KEYWORDS**

utf, unicode, toupper, tolower, totitle, case

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1997 by Sun Microsystems, Inc.
NAME
Tcl_LinkVar, Tcl_UnlinkVar, Tcl_UpdateLinkedVar - link Tcl variable to C variable

SYNOPSIS
#include <tcl.h>
int
Tcl_LinkVar(interp, varName, addr, type)
Tcl_UnlinkVar(interp, varName)
Tcl_UpdateLinkedVar(interp, varName)

ARGUMENTS

DESCRIPTION

TCL_LINK_INT
TCL_LINK_UINT
TCL_LINK_CHAR
TCL_LINK_UCHAR
TCL_LINK_SHORT
TCL_LINK_USHORT
TCL_LINK_LONG
TCL_LINK_ULONG
TCL_LINK_DOUBLE
TCL_LINK_FLOAT
TCL_LINK_WIDE_INT
TCL_LINK_WIDE_UINT
TCL_LINK_BOOLEAN
TCL_LINK_STRING

KEYWORDS
### SYNOPSIS

```c
#include <tcl.h>

int Tcl_LinkVar(interp, varName, addr, type)
Tcl_UnlinkVar(interp, varName)
Tcl_UpdateLinkedVar(interp, varName)
```

### ARGUMENTS

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp</em> (in)</td>
<td>Interpreter that contains <em>varName</em>. Also used by <strong>Tcl_LinkVar</strong> to return error messages.</td>
</tr>
<tr>
<td>const char <em>varName</em> (in)</td>
<td>Name of global variable.</td>
</tr>
<tr>
<td>char <em>addr</em> (in)</td>
<td>Address of C variable that is to be linked to <em>varName</em>.</td>
</tr>
</tbody>
</table>
to make Tcl variable read-only.

DESCRIPTION

**Tcl_LinkVar** uses variable traces to keep the Tcl variable named by `varName` in sync with the C variable at the address given by `addr`. Whenever the Tcl variable is read the value of the C variable will be returned, and whenever the Tcl variable is written the C variable will be updated to have the same value. **Tcl_LinkVar** normally returns **TCL_OK**; if an error occurs while setting up the link (e.g. because `varName` is the name of array) then **TCL_ERROR** is returned and the interpreter's result contains an error message.

The **type** argument specifies the type of the C variable, and must have one of the following values, optionally OR'ed with **TCL_LINK_READ_ONLY**:

**TCL_LINK_INT**

The C variable is of type **int**. Any value written into the Tcl variable must have a proper integer form acceptable to **Tcl_GetIntFromObj**; attempts to write non-integer values into `varName` will be rejected with Tcl errors.

**TCL_LINK_UINT**

The C variable is of type **unsigned int**. Any value written into the Tcl variable must have a proper unsigned integer form acceptable to **Tcl_GetWideIntFromObj** and in the platform's defined range for the **unsigned int** type; attempts to write non-integer values (or values outside the range) into `varName` will be rejected with Tcl errors.

**TCL_LINK_CHAR**

The C variable is of type **char**. Any value written into the Tcl variable must have a proper integer form acceptable to **Tcl_GetIntFromObj** and be in the range of the **char** datatype; attempts to write non-integer or out-of-range values into `varName`
will be rejected with Tcl errors.

**TCL\_LINK\_UCHAR**

The C variable is of type **unsigned char**. Any value written into the Tcl variable must have a proper unsigned integer form acceptable to **Tcl\_GetIntFromObj** and in the platform's defined range for the **unsigned char** type; attempts to write non-integer values (or values outside the range) into **varName** will be rejected with Tcl errors.

**TCL\_LINK\_SHORT**

The C variable is of type **short**. Any value written into the Tcl variable must have a proper integer form acceptable to **Tcl\_GetIntFromObj** and be in the range of the **short** datatype; attempts to write non-integer or out-of-range values into **varName** will be rejected with Tcl errors.

**TCL\_LINK\_USHORT**

The C variable is of type **unsigned short**. Any value written into the Tcl variable must have a proper unsigned integer form acceptable to **Tcl\_GetIntFromObj** and in the platform's defined range for the **unsigned short** type; attempts to write non-integer values (or values outside the range) into **varName** will be rejected with Tcl errors.

**TCL\_LINK\_LONG**

The C variable is of type **long**. Any value written into the Tcl variable must have a proper integer form acceptable to **Tcl\_GetLongFromObj**; attempts to write non-integer or out-of-range values into **varName** will be rejected with Tcl errors.

**TCL\_LINK\_ULONG**

The C variable is of type **unsigned long**. Any value written into the Tcl variable must have a proper unsigned integer form acceptable to **Tcl\_GetWideIntFromObj** and in the platform's defined range for the **unsigned long** type; attempts to write non-integer values (or values outside the range) into **varName** will be rejected with Tcl errors.
The C variable is of type double. Any value written into the Tcl variable must have a proper real form acceptable to \texttt{Tcl\_GetDoubleFromObj}; attempts to write non-real values into \texttt{varName} will be rejected with Tcl errors.

The C variable is of type float. Any value written into the Tcl variable must have a proper real form acceptable to \texttt{Tcl\_GetDoubleFromObj} and must be within the range acceptable for a float; attempts to write non-real values (or values outside the range) into \texttt{varName} will be rejected with Tcl errors.

The C variable is of type \texttt{Tcl\_WideInt} (which is an integer type at least 64-bits wide on all platforms that can support it.) Any value written into the Tcl variable must have a proper integer form acceptable to \texttt{Tcl\_GetWideIntFromObj}; attempts to write non-integer values into \texttt{varName} will be rejected with Tcl errors.

The C variable is of type \texttt{Tcl\_WideUInt} (which is an unsigned integer type at least 64-bits wide on all platforms that can support it.) Any value written into the Tcl variable must have a proper unsigned integer form acceptable to \texttt{Tcl\_GetWideIntFromObj} (it will be cast to unsigned); attempts to write non-integer values into \texttt{varName} will be rejected with Tcl errors.

The C variable is of type int. If its value is zero then it will read from Tcl as “0”; otherwise it will read from Tcl as “1”. Whenever \texttt{varName} is modified, the C variable will be set to a 0 or 1 value. Any value written into the Tcl variable must have a proper boolean form acceptable to \texttt{Tcl\_GetBooleanFromObj}; attempts to write non-boolean values into \texttt{varName} will be rejected with Tcl errors.

The C variable is of type char *. If its value is not NULL then it must
be a pointer to a string allocated with \texttt{Tcl\_Alloc} or \texttt{ckalloc}. Whenever the Tcl variable is modified the current C string will be freed and new memory will be allocated to hold a copy of the variable’s new value. If the C variable contains a NULL pointer then the Tcl variable will read as “NULL”.

If the \texttt{TCL\_LINK\_READ\_ONLY} flag is present in \texttt{type} then the variable will be read-only from Tcl, so that its value can only be changed by modifying the C variable. Attempts to write the variable from Tcl will be rejected with errors.

\texttt{Tcl\_UnlinkVar} removes the link previously set up for the variable given by \texttt{varName}. If there does not exist a link for \texttt{varName} then the procedure has no effect.

\texttt{Tcl\_UpdateLinkedVar} may be invoked after the C variable has changed to force the Tcl variable to be updated immediately. In many cases this procedure is not needed, since any attempt to read the Tcl variable will return the latest value of the C variable. However, if a trace has been set on the Tcl variable (such as a Tk widget that wishes to display the value of the variable), the trace will not trigger when the C variable has changed. \texttt{Tcl\_UpdateLinkedVar} ensures that any traces on the Tcl variable are invoked.

\begin{keywords}
boolean, integer, link, read-only, real, string, traces, variable
\end{keywords}
NAME
Tcl_ExprLong, Tcl_ExprDouble, Tcl_ExprBoolean, Tcl_ExprString - evaluate an expression

SYNOPSIS
#include <tcl.h>
int
Tcl_ExprLong(interp, expr, longPtr)
int
Tcl_ExprDouble(interp, expr, doublePtr)
int
Tcl_ExprBoolean(interp, expr, booleanPtr)
int
Tcl_ExprString(interp, expr)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS
int
Tcl_ExprString(interp, expr)

ARGUMENTS

Tcl_Interp *interp (in)  Interpreter in whose context to evaluate expr.

const char *expr (in)  Expression to be evaluated.

long *longPtr (out)  Pointer to location in which to store the integer value of the expression.

int *doublePtr (out)  Pointer to location in which to store the floating-point value of the expression.

int *booleanPtr (out)  Pointer to location in which to store the 0/1 boolean value of the expression.

DESCRIPTION

These four procedures all evaluate the expression given by the expr argument and return the result in one of four different forms. The expression can have any of the forms accepted by the expr command. Note that these procedures have been largely replaced by the object-based procedures Tcl_ExprLongObj, Tcl_ExprDoubleObj, Tcl_ExprBooleanObj, and Tcl_ExprObj. Those object-based procedures evaluate an expression held in a Tcl object instead of a string. The object argument can retain an internal representation that is more efficient to execute.

The interp argument refers to an interpreter used to evaluate the
expression (e.g. for variables and nested Tcl commands) and to return error information.

For all of these procedures the return value is a standard Tcl result: **TCL_OK** means the expression was successfully evaluated, and **TCL_ERROR** means that an error occurred while evaluating the expression. If **TCL_ERROR** is returned then the interpreter's result will hold a message describing the error. If an error occurs while executing a Tcl command embedded in the expression then that error will be returned.

If the expression is successfully evaluated, then its value is returned in one of four forms, depending on which procedure is invoked. **Tcl_ExprLong** stores an integer value at *longPtr*. If the expression's actual value is a floating-point number, then it is truncated to an integer. If the expression's actual value is a non-numeric string then an error is returned.

**Tcl_ExprDouble** stores a floating-point value at *doublePtr*. If the expression's actual value is an integer, it is converted to floating-point. If the expression's actual value is a non-numeric string then an error is returned.

**Tcl_ExprBoolean** stores a 0/1 integer value at *booleanPtr*. If the expression's actual value is an integer or floating-point number, then they store 0 at *booleanPtr* if the value was zero and 1 otherwise. If the expression's actual value is a non-numeric string then it must be one of the values accepted by **Tcl_GetBoolean** such as “yes” or “no”, or else an error occurs.

**Tcl_ExprString** returns the value of the expression as a string stored in the interpreter's result.

**SEE ALSO**

Tcl_ExprLongObj, Tcl_ExprDoubleObj, Tcl_ExprBooleanObj, Tcl_ExprObj
NAME
Tcl_ExprLongObj, Tcl_ExprDoubleObj, Tcl_ExprBooleanObj, Tcl_ExprObj - evaluate an expression

SYNOPSIS
#include <tcl.h>
int Tcl_ExprLongObj(interp, objPtr, longPtr)
int Tcl_ExprDoubleObj(interp, objPtr, doublePtr)
int Tcl_ExprBooleanObj(interp, objPtr, booleanPtr)
int Tcl_ExprObj(interp, objPtr, resultPtrPtr)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS
int
Tcl_ExprObj(interp, objPtr, resultPtrPtr)

ARGUMENTS

Tcl_Interp *interp (in)  
Interpreter in whose context to evaluate objPtr.

Tcl_Obj *objPtr (in)  
Pointer to an object containing the expression to evaluate.

long *longPtr (out)  
Pointer to location in which to store the integer value of the expression.

int *doublePtr (out)  
Pointer to location in which to store the floating-point value of the expression.

int *booleanPtr (out)  
Pointer to location in which to store the 0/1 boolean value of the expression.

Tcl_Obj **resultPtrPtr (out)  
Pointer to location in which to store a pointer to the object that is the result of the expression.

DESCRIPTION

These four procedures all evaluate an expression, returning the result in one of four different forms. The expression is given by the objPtr argument, and it can have any of the forms accepted by the expr command.
The *interp* argument refers to an interpreter used to evaluate the expression (e.g. for variables and nested Tcl commands) and to return error information.

For all of these procedures the return value is a standard Tcl result: **TCL_OK** means the expression was successfully evaluated, and **TCL_ERROR** means that an error occurred while evaluating the expression. If **TCL_ERROR** is returned, then a message describing the error can be retrieved using **Tcl_GetObjResult**. If an error occurs while executing a Tcl command embedded in the expression then that error will be returned.

If the expression is successfully evaluated, then its value is returned in one of four forms, depending on which procedure is invoked. **Tcl_ExprLongObj** stores an integer value at *longPtr. If the expression's actual value is a floating-point number, then it is truncated to an integer. If the expression's actual value is a non-numeric string then an error is returned.

**Tcl_ExprDoubleObj** stores a floating-point value at *doublePtr. If the expression's actual value is an integer, it is converted to floating-point. If the expression's actual value is a non-numeric string then an error is returned.

**Tcl_ExprBooleanObj** stores a 0/1 integer value at *booleanPtr. If the expression's actual value is an integer or floating-point number, then they store 0 at *booleanPtr if the value was zero and 1 otherwise. If the expression's actual value is a non-numeric string then it must be one of the values accepted by **Tcl_GetBoolean** such as “yes” or “no”, or else an error occurs.

If **Tcl_ExprObj** successfully evaluates the expression, it stores a pointer to the Tcl object containing the expression's value at *resultPtrPtr. In this case, the caller is responsible for calling **Tcl_DecrRefCount** to decrement the object's reference count when it is finished with the object.
**TclExprLong**, **TclExprDouble**, **TclExprBoolean**, **TclExprString**, **TclGetObjResult**

**KEYWORDS**

`boolean`, `double`, `evaluate`, `expression`, `integer`, `object`, `string`
NAME
Tcl_UpVar, Tcl_UpVar2 - link one variable to another

SYNOPSIS
#include <tcl.h>
int
Tcl_UpVar(interp, frameName, sourceName, destName, flags)
int
Tcl_UpVar2(interp, frameName, name1, name2, destName, flags)

ARGUMENTS

DESCRIPTION

KEYWORDS

NAME
Tcl_UpVar, Tcl_UpVar2 - link one variable to another

SYNOPSIS
#include <tcl.h>
int
Tcl_UpVar(interp, frameName, sourceName, destName, flags)
int
Tcl_UpVar2(interp, frameName, name1, name2, destName, flags)

ARGUMENTS

Tcl_Interp *interp (in)  Interpreter containing variables; also used for error reporting.

const char *frameName (in)  Identifies the stack frame
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>const char *sourceName</code> (in)</td>
<td>Name of source variable, in the frame given by <code>frameName</code>. May refer to a scalar variable or to an array variable with a parenthesized index.</td>
</tr>
<tr>
<td><code>const char *destName</code> (in)</td>
<td>Name of destination variable, which is to be linked to source variable so that references to <code>destName</code> refer to the other variable. Must not currently exist except as an upvar-ed variable.</td>
</tr>
<tr>
<td><code>int flags</code> (in)</td>
<td>One of <code>TCL_GLOBAL_ONLY</code>, <code>TCL_NAMESPACE_ONLY</code> or 0; if non-zero, then <code>destName</code> is a global or namespace variable; otherwise it is local to the current procedure (or current namespace if no procedure is active).</td>
</tr>
<tr>
<td><code>const char *name1</code> (in)</td>
<td>First part of source variable's name (scalar name, or name of array without array index).</td>
</tr>
</tbody>
</table>
const char *name2 (in)  If source variable is an element of an array, gives the index of the element. For scalar source variables, is NULL.

DESCRIPTION

Tcl_UpVar and Tcl_UpVar2 provide the same functionality as the upvar command: they make a link from a source variable to a destination variable, so that references to the destination are passed transparently through to the source. The name of the source variable may be specified either as a single string such as xyx or a(24) (by calling Tcl_UpVar) or in two parts where the array name has been separated from the element name (by calling Tcl_UpVar2). The destination variable name is specified in a single string; it may not be an array element.

Both procedures return either TCL_OK or TCL_ERROR, and they leave an error message in the interpreter's result if an error occurs.

As with the upvar command, the source variable need not exist; if it does exist, unsetting it later does not destroy the link. The destination variable may exist at the time of the call, but if so it must exist as a linked variable.

KEYWORDS

linked variable, upvar, variable

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
TCL_MEM_DEBUG - Compile-time flag to enable Tcl memory debugging

DESCRIPTION

When Tcl is compiled with TCL_MEM_DEBUG defined, a powerful set of memory debugging aids is included in the compiled binary. This includes C and Tcl functions which can aid with debugging memory leaks, memory allocation overruns, and other memory related errors.

ENABLING MEMORY DEBUGGING

To enable memory debugging, Tcl should be recompiled from scratch with TCL_MEM_DEBUG defined (e.g. by passing the --enable-symbols=mem flag to the configure script when building). This will also compile in a non-stub version of Tcl_InitMemory to add the memory command to Tcl.

TCL_MEM_DEBUG must be either left defined for all modules or undefined for all modules that are going to be linked together. If they are not, link errors will occur, with either Tcl_DbCkfree and Tcl_DbCkalloc or Tcl_Ckalloc and Tcl_Ckfree being undefined.

Once memory debugging support has been compiled into Tcl, the C functions Tcl_ValidateAllMemory, and Tcl_DumpActiveMemory, and the Tcl memory command can be used to validate and examine memory usage.

GUARD ZONES
When memory debugging is enabled, whenever a call to `ckalloc` is made, slightly more memory than requested is allocated so the memory debugging code can keep track of the allocated memory, and eight-byte “guard zones” are placed in front of and behind the space that will be returned to the caller. (The sizes of the guard zones are defined by the C `#define LOW_GUARD_SIZE` and `#define HIGH_GUARD_SIZE` in the file `generic/tclCkalloc.c` — it can be extended if you suspect large overwrite problems, at some cost in performance.) A known pattern is written into the guard zones and, on a call to `ckfree`, the guard zones of the space being freed are checked to see if either zone has been modified in any way. If one has been, the guard bytes and their new contents are identified, and a “low guard failed” or “high guard failed” message is issued. The “guard failed” message includes the address of the memory packet and the file name and line number of the code that called `ckfree`. This allows you to detect the common sorts of one-off problems, where not enough space was allocated to contain the data written, for example.

### DEBUGGING DIFFICULT MEMORY CORRUPTION PROBLEMS

Normally, Tcl compiled with memory debugging enabled will make it easy to isolate a corruption problem. Turning on memory validation with the memory command can help isolate difficult problems. If you suspect (or know) that corruption is occurring before the Tcl interpreter comes up far enough for you to issue commands, you can set `MEM_VALIDATE` define, recompile `tclCkalloc.c` and rebuild Tcl. This will enable memory validation from the first call to `ckalloc`, again, at a large performance impact.

If you are desperate and validating memory on every call to `ckalloc` and `ckfree` is not enough, you can explicitly call `Tcl_ValidateAllMemory` directly at any point. It takes a `char *` and an `int` which are normally the filename and line number of the caller, but they can actually be anything you want. Remember to remove the calls after you find the problem.

### SEE ALSO

- `ckalloc`
- `memory`
- `Tcl_ValidateAllMemory`
- `Tcl_DumpActiveMemory`
memory, debug
Tcl_WrongNumArgs - generate standard error message for wrong number of arguments

**SYNOPSIS**

`#include <tcl.h>`

Tcl_WrongNumArgs(interp, objc, objv, message)

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp interp</strong> (in)</td>
<td>Interpreter in which error will be reported: error message gets stored in its result object.</td>
</tr>
<tr>
<td>int <strong>objc</strong> (in)</td>
<td>Number of leading arguments from <strong>objv</strong> to include in error message.</td>
</tr>
<tr>
<td>Tcl_Obj *const <strong>objv[]</strong> (in)</td>
<td>Arguments to command that had the wrong number of arguments.</td>
</tr>
<tr>
<td>const char *message (in)</td>
<td>Additional error information to print after leading arguments from <strong>objv</strong>. This typically gives the acceptable syntax of the command. This</td>
</tr>
</tbody>
</table>
argument may be NULL.

**DESCRIPTION**

**Tcl_WrongNumArgs** is a utility procedure that is invoked by command procedures when they discover that they have received the wrong number of arguments. **Tcl_WrongNumArgs** generates a standard error message and stores it in the result object of interp. The message includes the *objc* initial elements of *objv* plus *message*. For example, if *objv* consists of the values *foo* and *bar*, *objc* is 1, and *message* is "fileName count" then *interp*'s result object will be set to the following string:

```
wrong # args: should be "foo fileName count"
```

If *objc* is 2, the result will be set to the following string:

```
wrong # args: should be "foo bar fileName count"
```

*objc* is usually 1, but may be 2 or more for commands like *string* and the Tk widget commands, which use the first argument as a subcommand.

Some of the objects in the *objv* array may be abbreviations for a subcommand. The command **Tcl_GetIndexFromObj** will convert the abbreviated string object into an *indexObject*. If an error occurs in the parsing of the subcommand we would like to use the full subcommand name rather than the abbreviation. If the **Tcl_WrongNumArgs** command finds any *indexObjects* in the *objv* array it will use the full subcommand name in the error message instead of the abbreviated name that was originally passed in. Using the above example, let us assume that *bar* is actually an abbreviation for *barfly* and the object is now an *indexObject* because it was passed to **Tcl_GetIndexFromObj**. In this case the error message would be:
wrong # args: should be "foo barfly fileName count"

SEE ALSO

Tcl_GetIndexFromObj

KEYWORDS

command, error message, wrong number of arguments

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME
Tk_Alloc3DBorderFromObj, Tk_Get3DBorder,
Tk_Get3DBorderFromObj, Tk_Draw3DRectangle,
Tk_Fill3DRectangle, Tk_Draw3DPolygon, Tk_Fill3DPolygon,
Tk_3DVerticalBevel, Tk_3DHorizontalBevel,
Tk_SetBackgroundFromBorder, Tk_NameOf3DBorder,
Tk_3DBorderColor, Tk_3DBorderGC,
Tk_Free3DBorderFromObj, Tk_Free3DBorder - draw borders
with three-dimensional appearance

SYNOPSIS
#include <tk.h>
Tk_3DBorder
Tk_Alloc3DBorderFromObj(interp, tkwin, objPtr)
Tk_3DBorder
Tk_Get3DBorder(interp, tkwin, colorName)
Tk_3DBorder
Tk_Get3DBorderFromObj(tkwin, objPtr)
void
Tk_Draw3DRectangle(tkwin, drawable, border, x, y, width,
height, borderWidth, relief)
void
Tk_Fill3DRectangle(tkwin, drawable, border, x, y, width,
height, borderWidth, relief)
void
Tk_Draw3DPolygon(tkwin, drawable, border, pointPtr,
numPoints, polyBorderWidth, leftRelief)
void
Tk_Fill3DPolygon(tkwin, drawable, border, pointPtr,
numPoints, polyBorderWidth, leftRelief)
void
Tk_3DVerticalBevel(tkwin, drawable, border, x, y, width,
void Tk_3DHorizontalBevel(tkwin, drawable, border, x, y, width, height, leftIn, rightIn, topBevel, relief)
void Tk_SetBackgroundFromBorder(tkwin, border)
const char * Tk_NameOf3DBorder(border)
XColor * Tk_3DBorderColor(border)
GC * Tk_3DBorderGC(tkwin, border, which)
Tk_Free3DBorderFromObj(tkwin, objPtr)
Tk_Free3DBorder(border)

ARGUMENTS
DESCRIPTION
KEYWORDS

NAME

Tk_Alloc3DBorderFromObj, Tk_Get3DBorder,
Tk_Get3DBorderFromObj, Tk_Draw3DRectangle, Tk_Fill3DRectangle,
Tk_Draw3DPolygon, Tk_Fill3DPolygon, Tk_3DVerticalBevel,
Tk_3DHorizontalBevel, Tk_SetBackgroundFromBorder,
Tk_NameOf3DBorder, Tk_3DBorderColor, Tk_3DBorderGC,
Tk_Free3DBorderFromObj, Tk_Free3DBorder - draw borders with three-dimensional appearance

SYNOPSIS

#include <tk.h>
Tk_3DBorder
Tk_Alloc3DBorderFromObj(interp, tkwin, objPtr)
Tk_3DBorder
Tk_Get3DBorder(interp, tkwin, colorName)
Tk_3DBorder
Tk_Get3DBorderFromObj(tkwin, objPtr)
void
Tk_Draw3DRectangle(tkwin, drawable, border, x, y, width, height, borderWidth, relief)
void
Tk_Fill3DRectangle(tkwin, drawable, border, x, y, width, height, borderWidth, relief)
void
Tk_Draw3DPolygon(tkwin, drawable, border, pointPtr, numPoints, polyBorderWidth, leftRelief)
void
Tk_Fill3DPolygon(tkwin, drawable, border, pointPtr, numPoints, polyBorderWidth, leftRelief)
void
Tk_3DVerticalBevel(tkwin, drawable, border, x, y, width, height, leftBevel, relief)
void
Tk_3DHorizontalBevel(tkwin, drawable, border, x, y, width, height, leftIn, rightIn, topBevel, relief)
void
Tk_SetBackgroundFromBorder(tkwin, border)
const char *
Tk_NameOf3DBorder(border)
XColor *
Tk_3DBorderColor(border)
GC *
Tk_3DBorderGC(tkwin, border, which)
Tk_Free3DBorderFromObj(tkwin, objPtr)
Tk_Free3DBorder(border)

ARGUMENTS

Tcl_Interp *interp (in) Interpreter to use for error reporting.

Tk_Window tkwin (in) Token for window (for all procedures except Tk_Get3DBorder, must be the window for which the
Tcl_Obj *objPtr (in)  
Pointer to object whose value describes color corresponding to background (flat areas). Illuminated edges will be brighter than this and shadowed edges will be darker than this.

char *colorName (in)  
Same as objPtr except value is supplied as a string rather than an object.

Drawable drawable (in)  
X token for window or pixmap; indicates where graphics are to be drawn. Must either be the X window for tkwin or a pixmap with the same screen and depth as tkwin.

Tk_3DBorder border (in)  
Token for border previously allocated in call to Tk_Get3DBorder.

int x (in)  
X-coordinate of upper-left corner of rectangle describing border or bevel, in pixels.

int y (in)  
Y-coordinate of upper-left corner of rectangle describing border or bevel, in pixels.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int width (in)</code></td>
<td>Width of rectangle describing border or bevel, in pixels.</td>
</tr>
<tr>
<td><code>int height (in)</code></td>
<td>Height of rectangle describing border or bevel, in pixels.</td>
</tr>
<tr>
<td><code>int borderWidth (in)</code></td>
<td>Width of border in pixels. Positive means border is inside rectangle given by x, y, width, height, negative means border is outside rectangle.</td>
</tr>
<tr>
<td><code>int relief (in)</code></td>
<td>Indicates 3-D position of interior of object relative to exterior; should be TK_RELIEF_RAISED, TK_RELIEF_SUNKEN, TK_RELIEF_GROOVE, TK_RELIEF_SOLID, or TK_RELIEF_RIDGE (may also be TK_RELIEF_FLAT for Tk_Fill3DRectangle).</td>
</tr>
<tr>
<td>XPoint *pointPtr (in)</td>
<td>Pointer to array of points describing the set of vertices in a polygon. The polygon need not be closed (it will be closed automatically if it is not).</td>
</tr>
<tr>
<td><code>int numPoints (in)</code></td>
<td>Number of points at *pointPtr.</td>
</tr>
<tr>
<td><code>int polyBorderWidth (in)</code></td>
<td>Width of border in pixels. If</td>
</tr>
</tbody>
</table>
positive, border is drawn to left of trajectory given by `pointPtr`; if negative, border is drawn to right of trajectory. If `leftRelief` is `TK_RELIEF_GROOVE` or `TK_RELIEF_RIDGE` then the border is centered on the trajectory.

<table>
<thead>
<tr>
<th>int <code>leftRelief</code> (in)</th>
<th>Height of left side of polygon’s path relative to right. <code>TK_RELIEF_RAISED</code> means left side should appear higher and <code>TK_RELIEF_SUNKEN</code> means right side should appear higher; <code>TK_RELIEF_GROOVE</code> and <code>TK_RELIEF_RIDGE</code> mean the obvious things. For <code>Tk_Fill3DPolygon</code>, <code>TK_RELIEF_FLAT</code> may also be specified to indicate no difference in height.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>int <code>leftBevel</code> (in)</th>
<th>Non-zero means this bevel forms the left side of the object; zero means it forms the right side.</th>
</tr>
</thead>
</table>

| int `leftIn` (in) | Non-zero means that the left edge of the horizontal bevel angles in, so that the bottom of the edge is |
These procedures provide facilities for drawing window borders in a way that produces a three-dimensional appearance.

**Tk_Alloc3DBorderFromObj** allocates colors and Pixmaps needed to draw a border in the window given by the *tkwin* argument. The value of *objPtr* is a standard Tk color name that determines the border colors. The color indicated by *objPtr* will not actually be used in the border; it
indicates the background color for the window (i.e. a color for flat surfaces). The illuminated portions of the border will appear brighter than indicated by objPtr, and the shadowed portions of the border will appear darker than objPtr.

Tk_Alloc3DBorderFromObj returns a token that may be used in later calls to Tk_Draw3DRectangle. If an error occurs in allocating information for the border (e.g. a bogus color name was given) then NULL is returned and an error message is left in interp->result. If it returns successfully, Tk_Alloc3DBorderFromObj caches information about the return value in objPtr, which speeds up future calls to Tk_Alloc3DBorderFromObj with the same objPtr and tkwin.

Tk_Get3DBorder is identical to Tk_Alloc3DBorderFromObj except that the color is specified with a string instead of an object. This prevents Tk_Get3DBorder from caching the return value, so Tk_Get3DBorder is less efficient than Tk_Alloc3DBorderFromObj.

Tk_Get3DBorderFromObj returns the token for an existing border, given the window and color name used to create the border. Tk_Get3DBorderFromObj does not actually create the border; it must already have been created with a previous call to Tk_Alloc3DBorderFromObj or Tk_Get3DBorder. The return value is cached in objPtr, which speeds up future calls to Tk_Get3DBorderFromObj with the same objPtr and tkwin.

Once a border structure has been created, Tk_Draw3DRectangle may be invoked to draw the border. The tkwin argument specifies the window for which the border was allocated, and drawable specifies a window or pixmap in which the border is to be drawn. Drawable need not refer to the same window as tkwin, but it must refer to a compatible pixmap or window: one associated with the same screen and with the same depth as tkwin. The x, y, width, and height arguments define the bounding box of the border region within drawable (usually x and y are zero and width and height are the dimensions of the window), and borderWidth specifies the number of pixels actually occupied by the border. The relief argument indicates which of several three-dimensional effects is desired: TK_RELIEF_RAISED means that the
interior of the rectangle should appear raised relative to the exterior of
the rectangle, and TK_RELIEF_SUNKEN means that the interior
should appear depressed. TK_RELIEF_GROOVE and
TK_RELIEF_RIDGE mean that there should appear to be a groove or
ridge around the exterior of the rectangle.

Tk_Fill3DRectangle is somewhat like Tk_Draw3DRectangle except
that it first fills the rectangular area with the background color (one
corresponding to the color used to create border). Then it calls
Tk_Draw3DRectangle to draw a border just inside the outer edge of
the rectangular area. The argument relief indicates the desired effect
(TK_RELIEF_FLAT means no border should be drawn; all that happens
is to fill the rectangle with the background color).

The procedure Tk_Draw3DPolygon may be used to draw more
complex shapes with a three-dimensional appearance. The pointPtr and
numPoints arguments define a trajectory, polyBorderWidth indicates
how wide the border should be (and on which side of the trajectory to
draw it), and leftRelief indicates which side of the trajectory should
appear raised. Tk_Draw3DPolygon draws a border around the given
trajectory using the colors from border to produce a three-dimensional
appearance. If the trajectory is non-self-intersecting, the appearance
will be a raised or sunken polygon shape. The trajectory may be self-
intersecting, although it's not clear how useful this is.

Tk_Fill3DPolygon is to Tk_Draw3DPolygon what
Tk_Fill3DRectangle is to Tk_Draw3DRectangle: it fills the polygonal
area with the background color from border, then calls
Tk_Draw3DPolygon to draw a border around the area (unless
leftRelief is TK_RELIEF_FLAT; in this case no border is drawn).

The procedures Tk_3DVerticalBevel and Tk_3DHorizontalBevel
provide lower-level drawing primitives that are used by procedures such
as Tk_Draw3DRectangle. These procedures are also useful in their
own right for drawing rectilinear border shapes. Tk_3DVerticalBevel
draws a vertical beveled edge, such as the left or right side of a
rectangle, and Tk_3DHorizontalBevel draws a horizontal beveled
edge, such as the top or bottom of a rectangle. Each procedure takes x,
The \texttt{Tk\_3DVerticalBevel} just draws a rectangular region. \texttt{Tk\_3DHorizontalBevel} draws a trapezoidal region to generate mitered corners; it should be called after \texttt{Tk\_3DVerticalBevel} (otherwise \texttt{Tk\_3DVerticalBevel} will overwrite the mitering in the corner). The \texttt{leftIn} and \texttt{rightIn} arguments to \texttt{Tk\_3DHorizontalBevel} describe the mitering at the corners; a value of 1 means that the bottom edge of the trapezoid will be shorter than the top, 0 means it will be longer. For example, to draw a rectangular border the top bevel should be drawn with 1 for both \texttt{leftIn} and \texttt{rightIn}, and the bottom bevel should be drawn with 0 for both arguments.

The procedure \texttt{Tk\_SetBackgroundFromBorder} will modify the background pixel and/or pixmap of \texttt{tkwin} to produce a result compatible with \texttt{border}. For color displays, the resulting background will just be the color specified when \texttt{border} was created; for monochrome displays, the resulting background will be a light stipple pattern, in order to distinguish the background from the illuminated portion of the border.

Given a token for a border, the procedure \texttt{Tk\_NameOf3DBorder} will return the color name that was used to create the border.

The procedure \texttt{Tk\_3DBorderColor} returns the XColor structure that will be used for flat surfaces drawn for its \texttt{border} argument by procedures like \texttt{Tk\_Fill3DRectangle}. The return value corresponds to the color name that was used to create the border. The XColor, and its associated pixel value, will remain allocated as long as \texttt{border} exists.

The procedure \texttt{Tk\_3DBorderGC} returns one of the X graphics contexts that are used to draw the border. The argument \texttt{which} selects which one of the three possible GC’s: \texttt{TK\_3D\_FLAT\_GC} returns the context used for flat surfaces, \texttt{TK\_3D\_LIGHT\_GC} returns the context for light shadows, and \texttt{TK\_3D\_DARK\_GC} returns the context for dark shadows.

When a border is no longer needed, \texttt{Tk\_Free3DBorderFromObj} or
Tk_Free3DBorder should be called to release the resources associated with it. For Tk_Free3DBorderFromObj the border to release is specified with the window and color name used to create the border; for Tk_Free3DBorder the border to release is specified with the Tk_3DBorder token for the border. There should be exactly one call to Tk_Free3DBorderFromObj or Tk_Free3DBorder for each call to Tk_Alloc3DBorderFromObj or Tk_Get3DBorder.

**KEYWORDS**

3D, background, border, color, depressed, illumination, object, polygon, raised, shadow, three-dimensional effect
NAME
Tk_WindowId, Tk_Parent, Tk_Display, Tk_DisplayName, Tk_ScreenNumber, Tk_Screen, Tk_X, Tk_Y, Tk_Width, Tk_Height, Tk_Changes, Tk_Attributes, Tk_IsContainer, Tk_IsEmbedded, Tk_IsMapped, Tk_IsTopLevel, Tk_ReqWidth, Tk_ReqHeight, Tk_MinReqWidth, Tk_MinReqHeight, Tk_InternalBorderLeft, Tk_InternalBorderRight, Tk_InternalBorderTop, Tk_InternalBorderBottom, Tk_Visual, Tk_Depth, Tk_Colormap, Tk_Interp - retrieve information from Tk's local data structure

SYNOPSIS
#include <tk.h>
Window
Tk_WindowId(tkwin)
Tk_Window
Tk_Parent(tkwin)
Display *
Tk_Display(tkwin)
const char *
Tk_DisplayName(tkwin)
int
Tk_ScreenNumber(tkwin)
Screen *
Tk_Screen(tkwin)
int
Tk_X(tkwin)
int
Tk_Y(tkwin)
int
Tk_Width(tkwin)
int
Tk_Height(tkwin)
XWindowChanges *
Tk_Changes(tkwin)
XSetWindowAttributes *
Tk_Attributes(tkwin)
int
Tk_IsContainer(tkwin)
int
Tk_IsEmbedded(tkwin)
int
Tk_IsMapped(tkwin)
int
Tk_IsTopLevel(tkwin)
int
Tk_ReqWidth(tkwin)
int
Tk_ReqHeight(tkwin)
int
Tk_MinReqWidth(tkwin)
int
Tk_MinReqHeight(tkwin)
int
Tk_InternalBorderLeft(tkwin)
int
Tk_InternalBorderRight(tkwin)
int
Tk_InternalBorderTop(tkwin)
int
Tk_InternalBorderBottom(tkwin)
int
Tk_Visual(tkwin)
Visual *
Tk_Depth(tkwin)
Colormap
Tk_Colormap(tkwin)
Tcl_Interp *
Tk_Interp(tkwin)
ARGUMENTS

DESCRIPTION

KEYWORDS

NAME

Tk_WindowId, Tk_Parent, Tk_Display, Tk_DisplayName, Tk_ScreenNumber, Tk_Screen, Tk_X, Tk_Y, Tk_Width, Tk_Height, Tk_Changes, Tk_Attributes, Tk_IsContainer, Tk_IsEmbedded, Tk_IsMapped, Tk_IsTopLevel, Tk_ReqWidth, Tk_ReqHeight, Tk_MinReqWidth, Tk_MinReqHeight, Tk_InternalBorderLeft, Tk_InternalBorderRight, Tk_InternalBorderTop, Tk_InternalBorderBottom, Tk_Visual, Tk_Depth, Tk_Colormap, Tk_Interp - retrieve information from Tk's local data structure

SYNOPSIS

#include <tk.h>
Window
Tk_WindowId(tkwin)
Tk_Window
Tk_Parent(tkwin)
Display *
Tk_Display(tkwin)
const char *
Tk_DisplayName(tkwin)
int
Tk_ScreenNumber(tkwin)
Screen *
Tk_Screen(tkwin)
int
Tk_X(tkwin)
int
Tk_Y(tkwin)
int
Tk_Width(tkwin)
int
Tk_Height(tkwin)
XWindowChanges *
Tk_Changes(tkwin)
XSetWindowAttributes *
Tk_Attributes(tkwin)
int
Tk_IsContainer(tkwin)
int
Tk_IsEmbedded(tkwin)
int
Tk_IsMapped(tkwin)
int
Tk_IsTopLevel(tkwin)
int
Tk_ReqWidth(tkwin)
int
Tk_ReqHeight(tkwin)
int
Tk_MinReqWidth(tkwin)
int
Tk_MinReqHeight(tkwin)
int
Tk_InternalBorderLeft(tkwin)
int
Tk_InternalBorderRight(tkwin)
int
Tk_InternalBorderTop(tkwin)
int
Tk_InternalBorderBottom(tkwin)
Visual *
Tk_Visual(tkwin)
int
Tk_Depth(tkwin)
Colormap
Tk_Colormap(tkwin)
Tcl_Interp *
Tk_Interp(tkwin)
ARGUMENTS

Tk_Window tkwin (in) Token for window.

DESCRIPTION

Tk_WindowId and the other names listed above are all macros that return fields from Tk's local data structure for tkwin. None of these macros requires any interaction with the server; it is safe to assume that all are fast.

Tk_WindowId returns the X identifier for tkwin, or NULL if no X window has been created for tkwin yet.

Tk_Parent returns Tk's token for the logical parent of tkwin. The parent is the token that was specified when tkwin was created, or NULL for main windows.

Tk_Interp returns the Tcl interpreter associated with a tkwin or NULL if there is an error.

Tk_Display returns a pointer to the Xlib display structure corresponding to tkwin. Tk_DisplayName returns an ASCII string identifying tkwin's display. Tk_ScreenNumber returns the index of tkwin's screen among all the screens of tkwin's display. Tk_Screen returns a pointer to the Xlib structure corresponding to tkwin's screen.

Tk_X, Tk_Y, Tk_Width, and Tk_Height return information about tkwin's location within its parent and its size. The location information refers to the upper-left pixel in the window, or its border if there is one. The width and height information refers to the interior size of the window, not including any border. Tk_Changes returns a pointer to a structure containing all of the above information plus a few other fields.

Tk_Attributes returns a pointer to an XSetWindowAttributes structure describing all of the attributes of the tkwin's window, such as background pixmap, event mask, and so on (Tk keeps track of all this information as it is changed by the application). Note: it is essential that
applications use Tk procedures like `Tk_ResizeWindow` instead of X procedures like `XResizeWindow`, so that Tk can keep its data structures up-to-date.

`Tk_IsContainer` returns a non-zero value if `tkwin` is a container, and that some other application may be embedding itself inside `tkwin`.

`Tk_IsEmbedded` returns a non-zero value if `tkwin` is not a free-standing window, but rather is embedded in some other application.

`Tk_IsMapped` returns a non-zero value if `tkwin` is mapped and zero if `tkwin` is not mapped.

`Tk_IsTopLevel` returns a non-zero value if `tkwin` is a top-level window (its X parent is the root window of the screen) and zero if `tkwin` is not a top-level window.

`Tk_ReqWidth` and `Tk_ReqHeight` return information about the window's requested size. These values correspond to the last call to `Tk_GeometryRequest` for `tkwin`.

`Tk_MinReqWidth` and `Tk_MinReqHeight` return information about the window's minimum requested size. These values correspond to the last call to `Tk_SetMinimumRequestSize` for `tkwin`.

`Tk_InternalBorderLeft`, `Tk_InternalBorderRight`, `Tk_InternalBorderTop` and `Tk_InternalBorderBottom` return the width of one side of the internal border that has been requested for `tkwin`, or 0 if no internal border was requested. The return value is simply the last value passed to `Tk_SetInternalBorder` or `Tk_SetInternalBorderEx` for `tkwin`.

`Tk_Visual`, `Tk_Depth`, and `Tk_Colormap` return information about the visual characteristics of a window. `Tk_Visual` returns the visual type for the window, `Tk_Depth` returns the number of bits per pixel, and `Tk_Colormap` returns the current colormap for the window. The visual characteristics are normally set from the defaults for the window's screen, but they may be overridden by calling `Tk_SetWindowVisual`. 
KEYWORDS

attributes, colormap, depth, display, height, geometry manager, identifier, mapped, requested size, screen, top-level, visual, width, window, x, y
NAME
Tk_GetReliefFromObj, Tk_GetRelief, Tk_NameOfRelief - translate between strings and relief values

SYNOPSIS
#include <tk.h>
int
Tk_GetReliefFromObj(interp, objPtr, reliefPtr)
int
Tk_GetRelief(interp, name, reliefPtr)
const char *
Tk_NameOfRelief(relief)

ARGUMENTS
DESCRIPTION
KEYWORDS
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interp *interp</code> (in)</td>
<td>Interpreter to use for error reporting.</td>
</tr>
<tr>
<td><code>Tcl_Obj *objPtr</code> (in/out)</td>
<td>String value contains name of relief, one of “flat”, “groove”, “raised”, “ridge”, “solid”, or “sunken”; the internal rep will be modified to cache corresponding relief value.</td>
</tr>
<tr>
<td><code>char *string</code> (in)</td>
<td>Same as <code>objPtr</code> except description of relief is passed as a string.</td>
</tr>
<tr>
<td><code>int *reliefPtr</code> (out)</td>
<td>Pointer to location in which to store relief value corresponding to <code>objPtr</code> or <code>name</code>.</td>
</tr>
<tr>
<td><code>const char *name ()</code></td>
<td>Name of the relief.</td>
</tr>
<tr>
<td><code>int relief</code> (in)</td>
<td>Relief value (one of <code>TK_RELIEF_FLAT</code>, <code>TK_RELIEF_RAISED</code>, <code>TK_RELIEF_SUNKEN</code>, <code>TK_RELIEF_GROOVE</code>, <code>TK_RELIEF_SOLID</code>, or <code>TK_RELIEF_RIDGE</code>).</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

`Tk_GetReliefFromObj` places in `*reliefPtr` the relief value corresponding to the value of `objPtr`. This value will be one of `TK_RELIEF_FLAT`, `TK_RELIEF_RAISED`, `TK_RELIEF_SUNKEN`, `TK_RELIEF_GROOVE`, `TK_RELIEF_SOLID`, or `TK_RELIEF_RIDGE`. 
Under normal circumstances the return value is TCL_OK and interp is unused. If objPtr does not contain one of the valid relief names or an abbreviation of one of them, then TCL_ERROR is returned, *reliefPtr is unmodified, and an error message is stored in interp's result if interp is not NULL. Tk_GetReliefFromObj caches information about the return value in objPtr, which speeds up future calls to Tk_GetReliefFromObj with the same objPtr.

Tk_GetRelief is identical to Tk_GetReliefFromObj except that the description of the relief is specified with a string instead of an object. This prevents Tk_GetRelief from caching the return value, so Tk_GetRelief is less efficient than Tk_GetReliefFromObj.

Tk_NameOfRelief is the logical inverse of Tk_GetRelief. Given a relief value it returns the corresponding string (flat, raised, sunken, groove, solid, or ridge). If relief is not a legal relief value, then “unknown relief” is returned.

**KEYWORDS**

name, relief, string

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1998 Sun Microsystems, Inc.
NAME
Tk_ComputeTextLayout, Tk_FreeTextLayout,
Tk_DrawTextLayout, Tk_UnderlineTextLayout,
Tk_PointToChar, Tk_CharBbox, Tk_DistanceToTextLayout,
Tk_IntersectTextLayout, Tk_TextLayoutToPostscript - routines
to measure and display single-font, multi-line, justified text.

SYNOPSIS
#include <tk.h>
Tk_TextLayout

Tk_ComputeTextLayout(tkfont, string, numChars,  
wrapLength, justify, flags, widthPtr, heightPtr)  
void

Tk_FreeTextLayout(layout)  
void

Tk_DrawTextLayout(display, drawable, gc, layout, x, y,  
firstChar, lastChar)  
void

Tk_UnderlineTextLayout(display, drawable, gc, layout, x, y,  
underline)  
int

Tk_PointToChar(layout, x, y)  
int

Tk_CharBbox(layout, index, xPtr, yPtr, widthPtr, heightPtr)  
int

Tk_DistanceToTextLayout(layout, x, y)  
int

Tk_IntersectTextLayout(layout, x, y, width, height)  
void

Tk_TextLayoutToPostscript(interp, layout)

ARGUMENTS
DESCRIPTION
DISPLAY MODEL
KEYWORDS

NAME

Tk_ComputeTextLayout, Tk_FreeTextLayout, Tk_DrawTextLayout,
Tk_UnderlineTextLayout, Tk_PointToChar, Tk_CharBbox,
Tk_DistanceToTextLayout, Tk_IntersectTextLayout,
Tk_TextLayoutToPostscript - routines to measure and display single-
font, multi-line, justified text.

SYNOPSIS

#include <tk.h>
Tk_TextLayout
Tk_ComputeTextLayout(tkfont, string, numChars, wrapLength, justify,
flags, widthPtr, heightPtr)
void
Tk_FreeTextLayout(layout)
void
Tk_DrawTextLayout(display, drawable, gc, layout, x, y, firstChar,
lastChar)
void
Tk_UnderlineTextLayout(display, drawable, gc, layout, x, y, underline)
int
Tk_PointToChar(layout, x, y)
int
Tk_CharBbox(layout, index, xPtr, yPtr, widthPtr, heightPtr)
int
Tk_DistanceToTextLayout(layout, x, y)
int
Tk_IntersectTextLayout(layout, x, y, width, height)
void
Tk_TextLayoutToPostscript(interp, layout)

ARGUMENTS
<table>
<thead>
<tr>
<th>Tk_Font tkfont (in)</th>
<th>Font to use when constructing and displaying a text layout. The tkfont must remain valid for the lifetime of the text layout. Must have been returned by a previous call to Tk_GetFont.</th>
</tr>
</thead>
<tbody>
<tr>
<td>const char *string (in)</td>
<td>Potentially multi-line string whose dimensions are to be computed and stored in the text layout. The string must remain valid for the lifetime of the text layout.</td>
</tr>
<tr>
<td>int numChars (in)</td>
<td>The number of characters to consider from string. If numChars is less than 0, then assumes string is null terminated and uses Tcl_NumUtfChars to determine the length of string.</td>
</tr>
<tr>
<td>int wrapLength (in)</td>
<td>Longest permissible line length, in pixels. Lines in string will automatically be broken at word boundaries and wrapped when they reach this length. If wrapLength is too small for even a single character to fit on a line, it will be expanded to allow one character to fit on each</td>
</tr>
</tbody>
</table>
How to justify the lines in a multi-line text layout. Possible values are `TK_JUSTIFY_LEFT`, `TK_JUSTIFY_CENTER`, or `TK_JUSTIFY_RIGHT`. If the text layout only occupies a single line, then `justify` is irrelevant.

Various flag bits OR-ed together. `TK_IGNORE_TABS` means that tab characters should not be expanded to the next tab stop. `TK_IGNORE_NEWLINES` means that newline/return characters should not cause a line break. If either tabs or newlines/returns are ignored, then they will be treated as regular characters, being measured and displayed in a platform-dependent manner as described in `Tk_MeasureChars`, and will not have any special
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int *widthPtr (out)</td>
<td>If non-NULL, filled with either the width, in pixels, of the widest line in the text layout, or the width, in pixels, of the bounding box for the character specified by index.</td>
</tr>
<tr>
<td>int *heightPtr (out)</td>
<td>If non-NULL, filled with either the total height, in pixels, of all the lines in the text layout, or the height, in pixels, of the bounding box for the character specified by index.</td>
</tr>
<tr>
<td>Tk_TextLayout layout (in)</td>
<td>A token that represents the cached layout information about the single-font, multi-line, justified piece of text. This token is returned by Tk_ComputeTextLayout.</td>
</tr>
<tr>
<td>Display *display (in)</td>
<td>Display on which to draw.</td>
</tr>
<tr>
<td>Drawable drawable (in)</td>
<td>Window or pixmap in which to draw.</td>
</tr>
<tr>
<td>GC gc (in)</td>
<td>Graphics context to use for drawing text layout. The font selected in this GC must correspond to the tkfont used when constructing the text behaviors.</td>
</tr>
<tr>
<td><strong>int x, y (in)</strong></td>
<td>Point, in pixels, at which to place the upper-left hand corner of the text layout when it is being drawn, or the coordinates of a point (with respect to the upper-left hand corner of the text layout) to check against the text layout.</td>
</tr>
<tr>
<td><strong>int firstChar (in)</strong></td>
<td>The index of the first character to draw from the given text layout. The number 0 means to draw from the beginning.</td>
</tr>
<tr>
<td><strong>int lastChar (in)</strong></td>
<td>The index of the last character up to which to draw. The character specified by <code>lastChar</code> itself will not be drawn. A number less than 0 means to draw all characters in the text layout.</td>
</tr>
<tr>
<td><strong>int underline (in)</strong></td>
<td>Index of the single character to underline in the text layout, or a number less than 0 for no underline.</td>
</tr>
<tr>
<td><strong>int index (in)</strong></td>
<td>The index of the character whose bounding box is desired. The bounding box is computed with respect</td>
</tr>
</tbody>
</table>
These routines are for measuring and displaying single-font, multi-line, justified text. To measure and display simple single-font, single-line strings, refer to the documentation for `Tk_MeasureChars`. There is no programming interface in the core of Tk that supports multi-font, multi-line text; support for that behavior must be built on top of simpler layers. Note that unlike the lower level text display routines, the functions described here all operate on character-oriented lengths and indices rather than byte-oriented values. See the description of `Tcl_UtfAtIndex` for more details on converting between character and byte offsets.

The routines described here are built on top of the programming
interface described in the Tk.MeasureChars documentation. Tab characters and newline/return characters may be treated specially by these procedures, but all other characters are passed through to the lower level.

Tk.ComputeTextLayout computes the layout information needed to display a single-font, multi-line, justified string of text and returns a Tk.TextLayout token that holds this information. This token is used in subsequent calls to procedures such as Tk.DrawTextLayout, Tk.DistanceToTextLayout, and Tk.FreeTextLayout. The string and tkfont used when computing the layout must remain valid for the lifetime of this token.

Tk.FreeTextLayout is called to release the storage associated with layout when it is no longer needed. A layout should not be used in any other text layout procedures once it has been released.

Tk.DrawTextLayout uses the information in layout to display a single-font, multi-line, justified string of text at the specified location.

Tk.UnderlineTextLayout uses the information in layout to display an underline below an individual character. This procedure does not draw the text, just the underline. To produce natively underlined text, an underlined font should be constructed and used. All characters, including tabs, newline/return characters, and spaces at the ends of lines, can be underlined using this method. However, the underline will never be drawn outside of the computed width of layout; the underline will stop at the edge for any character that would extend partially outside of layout, and the underline will not be visible at all for any character that would be located completely outside of the layout.

Tk.PointToChar uses the information in layout to determine the character closest to the given point. The point is specified with respect to the upper-left hand corner of the layout, which is considered to be located at (0, 0). Any point whose y-value is less that 0 will be considered closest to the first character in the text layout; any point whose y-value is greater than the height of the text layout will be considered closest to the last character in the text layout. Any point
whose x-value is less than 0 will be considered closest to the first character on that line; any point whose x-value is greater than the width of the text layout will be considered closest to the last character on that line. The return value is the index of the character that was closest to the point. Given a layout with no characters, the value 0 will always be returned, referring to a hypothetical zero-width placeholder character.

Tk_CharBbox uses the information in layout to return the bounding box for the character specified by index. The width of the bounding box is the advance width of the character, and does not include any left or right bearing. Any character that extends partially outside of layout is considered to be truncated at the edge. Any character that would be located completely outside of layout is considered to be zero-width and pegged against the edge. The height of the bounding box is the line height for this font, extending from the top of the ascent to the bottom of the descent; information about the actual height of individual letters is not available. For measurement purposes, a layout that contains no characters is considered to contain a single zero-width placeholder character at index 0. If index was not a valid character index, the return value is 0 and *xPtr, *yPtr, *widthPtr, and *heightPtr are unmodified. Otherwise, if index did specify a valid, the return value is non-zero, and *xPtr, *yPtr, *widthPtr, and *heightPtr are filled with the bounding box information for the character. If any of xPtr, yPtr, widthPtr, or heightPtr are NULL, the corresponding value is not calculated or stored.

Tk_DistanceToTextLayout computes the shortest distance in pixels from the given point (x, y) to the characters in layout. Newline/return characters and non-displaying space characters that occur at the end of individual lines in the text layout are ignored for hit detection purposes, but tab characters are not. The return value is 0 if the point actually hits the layout. If the point did not hit the layout then the return value is the distance in pixels from the point to the layout.

Tk_IntersectTextLayout determines whether a layout lies entirely inside, entirely outside, or overlaps a given rectangle. Newline/return characters and non-displaying space characters that occur at the end of individual lines in the layout are ignored for intersection calculations. The return value is -1 if the layout is entirely outside of the rectangle, 0
if it overlaps, and 1 if it is entirely inside of the rectangle.

**Tk_TextLayoutToPostscript** outputs code consisting of a Postscript array of strings that represent the individual lines in *layout*. It is the responsibility of the caller to take the Postscript array of strings and add some Postscript function operate on the array to render each of the lines. The code that represents the Postscript array of strings is appended to `interp->result`.

---

**DISPLAY MODEL**

When measuring a text layout, space characters that occur at the end of a line are ignored. The space characters still exist and the insertion point can be positioned amongst them, but their additional width is ignored when justifying lines or returning the total width of a text layout. All end-of-line space characters are considered to be attached to the right edge of the line; this behavior is logical for left-justified text and reasonable for center-justified text, but not very useful when editing right-justified text. Spaces are considered variable width characters; the first space that extends past the edge of the text layout is clipped to the edge, and any subsequent spaces on the line are considered zero width and pegged against the edge. Space characters that occur in the middle of a line of text are not suppressed and occupy their normal space width.

Tab characters are not ignored for measurement calculations. If wrapping is turned on and there are enough tabs on a line, the next tab will wrap to the beginning of the next line. There are some possible strange interactions between tabs and justification; tab positions are calculated and the line length computed in a left-justified world, and then the whole resulting line is shifted so it is centered or right-justified, causing the tab columns not to align any more.

When wrapping is turned on, lines may wrap at word breaks (space or tab characters) or newline/returns. A dash or hyphen character in the middle of a word is not considered a word break. **Tk_ComputeTextLayout** always attempts to place at least one word on each line. If it cannot because the `wrapLength` is too small, the word will
be broken and as much as fits placed on the line and the rest on subsequent line(s). If \texttt{wrapLength} is so small that not even one character can fit on a given line, the \texttt{wrapLength} is ignored for that line and one character will be placed on the line anyhow. When wrapping is turned off, only newline/return characters may cause a line break.

When a text layout has been created using an underlined \texttt{tkfont}, then any space characters that occur at the end of individual lines, newlines/returns, and tabs will not be displayed underlined when \texttt{Tk\_DrawTextLayout} is called, because those characters are never actually drawn - they are merely placeholders maintained in the \texttt{layout}.

\section*{KEYWORDS}
\begin{itemize}
\item font
\end{itemize}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
NAME

Tk_GetRootCoords - Compute root-window coordinates of window

SYNOPSIS

#include <tk.h>
Tk_GetRootCoords(tkwin, xPtr, yPtr)

ARGUMENTS

Tk_Window tkwin (in) Token for window.

int *xPtr (out) Pointer to location in which to store root-window x-coordinate corresponding to left edge of tkwin's border.

int *yPtr (out) Pointer to location in which to store root-window y-coordinate corresponding to top edge of tkwin's border.

DESCRIPTION

This procedure scans through the structural information maintained by Tk to compute the root-window coordinates corresponding to the upper-left corner of tkwin's border. If tkwin has no border, then Tk_GetRootCoords returns the root-window coordinates
corresponding to location (0,0) in \texttt{tkwin}. \texttt{Tk\_GetRootCoords} is relatively efficient, since it does not have to communicate with the X server.

**KEYWORDS**

coordinates, root window

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tk_FontId, Tk_GetFontMetrics, Tk_PostscriptFontName - accessor functions for fonts

SYNOPSIS
#include <tk.h>
Font
Tk_FontId(tkfont)
Tk_GetFontMetrics(tkfont, fmPtr)
int
Tk_PostscriptFontName(tkfont, dsPtr)

ARGUMENTS
Tk_Font tkfont (in)  
Opaque font token being queried. Must have been returned by a previous call to Tk_GetFont.

Tk_FontMetrics *fmPtr (out)  
Pointer to structure in which the font metrics for tkfont will be stored. See DATA STRUCTURES below for details.

Tcl_DString *dsPtr (out)  
Pointer to an initialized Tcl_DString to which the name of the Postscript font that corresponds to tkfont will be appended.

**DESCRIPTION**

Given a tkfont, Tk_FontId returns the token that should be selected into an XGCValues structure in order to construct a graphics context that can be used to draw text in the specified font.

Tk_GetFontMetrics computes the ascent, descent, and linespace of the tkfont in pixels and stores those values in the structure pointer to by fmPtr. These values can be used in computations such as to space multiple lines of text, to align the baselines of text in different fonts, and to vertically align text in a given region. See the documentation for the font command for definitions of the terms ascent, descent, and linespace, used in font metrics.

Tk_PostscriptFontName maps a tkfont to the corresponding Postscript font name that should be used when printing. The return value is the size in points of the tkfont and the Postscript font name is appended to dsPtr. DsPtr must refer to an initialized Tcl_DString. Given a “reasonable” Postscript printer, the following screen font families should
print correctly:


Any other font families may not print correctly because the computed Postscript font name may be incorrect or not exist on the printer.

**DATA STRUCTURES**

The **Tk_FontMetrics** data structure is used by **Tk_GetFontMetrics** to return information about a font and is defined as follows:

```c
typedef struct Tk_FontMetrics {
    int ascent;
    int descent;
    int linespace;
} Tk_FontMetrics;
```

The *ascent* field is the amount in pixels that the tallest letter sticks up above the baseline, plus any extra blank space added by the designer of the font.

The *descent* is the largest amount in pixels that any letter sticks below the baseline, plus any extra blank space added by the designer of the font.

The *linespace* is the sum of the ascent and descent. How far apart two lines of text in the same font should be placed so that none of the characters in one line overlap any of the characters in the other line.

**SEE ALSO**

font, MeasureChar
KEYWORDS

font, measurement, Postscript

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
NAME
Tk_GetPixelsFromObj, Tk_GetPixels, Tk_GetMMFromObj,
Tk_GetScreenMM - translate between strings and screen units

SYNOPSIS
#include <tk.h>
int Tk_GetPixelsFromObj(interp, tkwin, objPtr, intPtr)
int Tk_GetPixels(interp, tkwin, string, intPtr)
int Tk_GetMMFromObj(interp, tkwin, objPtr, doublePtr)
int Tk_GetScreenMM(interp, tkwin, string, doublePtr)

ARGUMENTS
<none>

DESCRIPTION
  c
  i
  m
  p

KEYWORDS

NAME
Tk_GetPixelsFromObj, Tk_GetPixels, Tk_GetMMFromObj,
Tk_GetScreenMM - translate between strings and screen units

SYNOPSIS
#include <tk.h>
int Tk_GetPixelsFromObj(interp, tkwin, objPtr, intPtr)
The functions `Tk_GetPixels()`, `Tk_GetMMFromObj()` and `Tk_GetScreenMM()` are used to convert distances specified in the `string` parameter to pixels and millimeters, respectively. These conversions are based on the `window` specified by `tkwin`, whose screen geometry determines the conversion between absolute units and pixels.

### Arguments

<table>
<thead>
<tr>
<th><strong>Argument</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interp *interp (in)</code></td>
<td>Interpreter to use for error reporting.</td>
</tr>
<tr>
<td><code>Tk_Window tkwin (in)</code></td>
<td>Window whose screen geometry determines the conversion between absolute units and pixels.</td>
</tr>
<tr>
<td><code>Tcl_Obj *objPtr (in/out)</code></td>
<td>String value specifies a distance on the screen; internal rep will be modified to cache converted distance.</td>
</tr>
<tr>
<td><code>const char *string (in)</code></td>
<td>Same as <code>objPtr</code> except specification of distance is passed as a string.</td>
</tr>
<tr>
<td><code>int *intPtr (out)</code></td>
<td>Pointer to location in which to store converted distance in pixels.</td>
</tr>
<tr>
<td><code>double *doublePtr (out)</code></td>
<td>Pointer to location in which to store converted distance in millimeters.</td>
</tr>
</tbody>
</table>

### Description

The `Tk_GetPixels()` function retrieves the converted distance in pixels, the `Tk_GetMMFromObj()` function retrieves the converted distance in millimeters, and the `Tk_GetScreenMM()` function retrieves the converted distance relative to the screen geometry of the `tkwin` window.
These procedures take as argument a specification of distance on the screen (\texttt{objPtr or string}) and compute the corresponding distance either in integer pixels or floating-point millimeters. In either case, \texttt{objPtr} or \texttt{string} specifies a screen distance as a floating-point number followed by one of the following characters that indicates units:

\begin{itemize}
  \item \texttt{<none>} : The number specifies a distance in pixels.
  \item \texttt{c} : The number specifies a distance in centimeters on the screen.
  \item \texttt{i} : The number specifies a distance in inches on the screen.
  \item \texttt{m} : The number specifies a distance in millimeters on the screen.
  \item \texttt{p} : The number specifies a distance in printer's points (1/72 inch) on the screen.
\end{itemize}

\texttt{Tk\_GetPixelsFromObj} converts the value of \texttt{objPtr} to the nearest even number of pixels and stores that value at \texttt{*intPtr}. It returns \texttt{TCL\_OK} under normal circumstances. If an error occurs (e.g. \texttt{objPtr} contains a number followed by a character that is not one of the ones above) then \texttt{TCL\_ERROR} is returned and an error message is left in \texttt{interp}'s result if \texttt{interp} is not NULL. \texttt{Tk\_GetPixelsFromObj} caches information about the return value in \texttt{objPtr}, which speeds up future calls to \texttt{Tk\_GetPixelsFromObj} with the same \texttt{objPtr}.

\texttt{Tk\_GetPixels} is identical to \texttt{Tk\_GetPixelsFromObj} except that the screen distance is specified with a string instead of an object. This prevents \texttt{Tk\_GetPixels} from caching the return value, so \texttt{Tk\_GetAnchor} is less efficient than \texttt{Tk\_GetPixelsFromObj}.

\texttt{Tk\_GetMMFromObj} and \texttt{Tk\_GetScreenMM} are similar to \texttt{Tk\_GetPixelsFromObj} and \texttt{Tk\_GetPixels} (respectively) except that
they convert the screen distance to millimeters and store a double-
precision floating-point result at *doublePtr.

**KEYWORDS**

centimeters, convert, inches, millimeters, pixels, points, screen units

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1998 Sun Microsystems, Inc.
NAME
Tk_GetColormap, Tk_PreserveColormap, Tk_FreeColormap - allocate and free colormaps

SYNOPSIS
#include <tk.h>
Colormap
Tk_GetColormap(interp, tkwin, string)
Tk_PreserveColormap(display, colormap)
Tk_FreeColormap(display, colormap)

ARGUMENTS
Tcl_Interp *interp (in) Interpreter to use for error reporting.
Tk_Window tkwin (in) Token for window in which
const char *string (in)  
Selects a colormap: either new or the name of a window with the same screen and visual as tkwin.

Display *display (in)  
Display for which colormap was allocated.

Colormap colormap (in)  
Colormap to free or preserve; must have been returned by a previous call to Tk_GetColormap or Tk_GetVisual.

DESCRIPTION

These procedures are used to manage colormaps. Tk_GetColormap returns a colormap suitable for use in tkwin. If its string argument is new then a new colormap is created; otherwise string must be the name of another window with the same screen and visual as tkwin, and the colormap from that window is returned. If string does not make sense, or if it refers to a window on a different screen from tkwin or with a different visual than tkwin, then Tk_GetColormap returns None and leaves an error message in interp's result.

Tk_PreserveColormap increases the internal reference count for a colormap previously returned by Tk_GetColormap, which allows the colormap to be stored in several locations without knowing which order they will be released.

Tk_FreeColormap should be called when a colormap returned by Tk_GetColormap is no longer needed. Tk maintains a reference count for each colormap returned by Tk_GetColormap, so there should eventually be one call to Tk_FreeColormap for each call to Tk_GetColormap and each call to Tk_PreserveColormap. When a
colormap's reference count becomes zero, Tk releases the X colormap.

Tk_GetVisual and Tk_GetColormap work together, in that a new colormap created by Tk_GetVisual may later be returned by Tk_GetColormap. The reference counting mechanism for colormaps includes both procedures, so callers of Tk_GetVisual must also call Tk_FreeColormap to release the colormap. If Tk_GetColormap is called with a string value of new then the resulting colormap will never be returned by Tk_GetVisual; however, it can be used in other windows by calling Tk_GetColormap with the original window's name as string.

KEYWORDS

colormap, visual

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tk_GetScrollInfo, Tk_GetScrollInfoObj - parse arguments for scrolling commands

SYNOPSIS
#include <tk.h>
int Tk_GetScrollInfo(interp, argc, argv, dblPtr, intPtr)
int Tk_GetScrollInfoObj(interp, objc, objv, dblPtr, intPtr)

ARGUMENTS

Tk_Interp *interp (in) Interpreter to use for error reporting.
int argc (in) Number of strings in argv
# DESCRIPTION

**Tk_GetScrollInfo** parses the arguments expected by widget scrolling commands such as `xview` and `yview`. It receives the entire list of words that make up a widget command and parses the words starting with `argv[2]`. The words starting with `argv[2]` must have one of the following forms:

```c
const char *argv[] (in)
```

Argument strings. These represent the entire widget command, of which the first word is typically the widget name and the second word is typically `xview` or `yview`.

```c
int objc (in)
```

Number of `Tcl_Obj`'s in `objc` array.

```c
Tcl_Obj *const objv[] (in)
```

Argument objects. These represent the entire widget command, of which the first word is typically the widget name and the second word is typically `xview` or `yview`.

```c
double *dblPtr (out)
```

Filled in with fraction from `moveto` option, if any.

```c
int *intPtr (out)
```

Filled in with line or page count from `scroll` option, if any. The value may be negative.
**moveto** *fraction*
**scroll** *number units*
**scroll** *number pages*

Any of the **moveto**, **scroll**, **units**, and **pages** keywords may be abbreviated. If `argv` has the **moveto** form, **TK_SCROLL_MOVETO** is returned as result and `*dblPtr` is filled in with the *fraction* argument to the command, which must be a proper real value. If `argv` has the **scroll** form, **TK_SCROLL_UNITS** or **TK_SCROLL_PAGES** is returned and `*intPtr` is filled in with the *number* value, which must be a proper integer. If an error occurs in parsing the arguments, **TK_SCROLL_ERROR** is returned and an error message is left in `interp->result`.

**Tk_GetScrollInfoObj** is identical in function to **Tk_GetScrollInfo**. However, **Tk_GetScrollInfoObj** accepts Tcl_Obj style arguments, making it more appropriate for use with new development.

**KEYWORDS**

**parse**, **scrollbar**, **scrolling command**, **xview**, **yview**

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
Tk_CollapseMotionEvents, Tk_QueueWindowEvent - Add a window event to the Tcl event queue

SYNOPSIS

#include <tk.h>
int Tk_CollapseMotionEvents(display, collapse)
Tk_QueueWindowEvent(eventPtr, position)

ARGUMENTS

Display *display (in)  Display for which to control motion event collapsing.
int collapse (in)  Indicates whether motion events should be collapsed or not.
XEvent *eventPtr (in)  An event to add to the event queue.
Tcl_QueuePosition position (in)  Where to add the new event in the queue: TCL_QUEUE_TAIL, TCL_QUEUE_HEAD, or TCL_QUEUE_MARK.

DESCRIPTION
**Tk.QueueWindowEvent** places a window event on Tcl's internal event queue for eventual servicing. It creates a Tcl_Event structure, copies the event into that structure, and calls **Tcl.QueueEvent** to add the event to the queue. When the event is eventually removed from the queue it is processed just like all window events.

When multiple motion events are received for the same window in rapid succession, they are collapsed by default. This behavior can be controlled with **Tk.CollapseMotionEvents**. **Tk.CollapseMotionEvents** always returns the previous value for collapse behavior on the display.

The *position* argument to **Tk.QueueWindowEvent** has the same significance as for **Tcl.QueueEvent**; see the documentation for **Tcl.QueueEvent** for details.

---

**KEYWORDS**

callback, clock, handler, modal timeout, events

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME
Tk_AddOption - Add an option to the option database

SYNOPSIS
#include <tk.h>
void
Tk_AddOption(tkwin, name, value, priority)

ARGUMENTS

DESCRIPTION

KEYWORDS
int **priority** (in) Overall priority level to use for option.

### DESCRIPTION

This procedure is invoked to add an option to the database associated with *tkwin*'s main window. *Name* contains the option being specified and consists of names and/or classes separated by asterisks or dots, in the usual X format. *Value* contains the text string to associate with *name*; this value will be returned in calls to **Tk_GetOption**. *Priority* specifies the priority of the value; when options are queried using **Tk_GetOption**, the value with the highest priority is returned. *Priority* must be between 0 and **TK_MAX_PRIO**. Some common priority values are:

- **20**
  - Used for default values hard-coded into widgets.
- **40**
  - Used for options specified in application-specific startup files.
- **60**
  - Used for options specified in user-specific defaults files, such as **.Xdefaults**, resource databases loaded into the X server, or user-specific startup files.
- **80**
  - Used for options specified interactively after the application starts running.

### KEYWORDS

**class**, **name**, **option**, **add**

---

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
Tk_MeasureChars, Tk_TextWidth, Tk_DrawChars, Tk_UnderlineChars - routines to measure and display simple single-line strings.

SYNOPSIS
#include <tk.h>
int
Tk_MeasureChars(tkfont, string, numBytes, maxPixels, flags, lengthPtr)

int
Tk_TextWidth(tkfont, string, numBytes)

Tk_DrawChars(display, drawable, gc, tkfont, string, numBytes, x, y)

Tk_UnderlineChars(display, drawable, gc, tkfont, string, x, y, firstByte, lastByte)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS
**Tk_TextWidth(tkfont, string, numBytes)**

**Tk_DrawChars(display, drawable, gc, tkfont, string, numBytes, x, y)**

**Tk_UnderlineChars(display, drawable, gc, tkfont, string, x, y, firstByte, lastByte)**

### ARGUMENTS

**Tk_Font tkfont (in)**

Token for font in which text is to be drawn or measured. Must have been returned by a previous call to *Tk_GetFont*.

**const char *string (in)**

Text to be measured or displayed. Need not be null terminated. Any non-printing meta-characters in the string (such as tabs, newlines, and other control characters) will be measured or displayed in a platform-dependent manner.

**int numBytes (in)**

The maximum number of bytes to consider when measuring or drawing *string*. Must be greater than or equal to 0.

**int maxPixels (in)**

If *maxPixels* is >= 0, it specifies the longest permissible line length in pixels. Characters from *string* are processed only until this many pixels have
been covered. If `maxPixels` is < 0, then the line length is unbounded and the `flags` argument is ignored.

| **int flags (in)** | Various flag bits OR-ed together:  
|                   | **TK_PARTIAL_OK** means include a character as long as any part of it fits in the length given by `maxPixels`; otherwise, a character must fit completely to be considered.  
|                   | **TK_WHOLE_WORDS** means stop on a word boundary, if possible. If **TK_AT_LEAST_ONE** is set, it means return at least one character even if no characters could fit in the length given by `maxPixels`. If **TK_AT_LEAST_ONE** is set and  
|                   | **TK_WHOLE_WORDS** is also set, it means that if not even one word fits on the line, return the first few letters of the word that did fit; if not even one letter of the word fit, then the first letter will still be returned. |

| **int *lengthPtr (out)** | Filled with the number of pixels occupied by the number of characters |
returned as the result of `Tk_MeasureChars`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display <em>display</em></td>
<td>Display on which to draw.</td>
</tr>
<tr>
<td>Drawable drawable</td>
<td>Window or pixmap in which to draw.</td>
</tr>
<tr>
<td>GC gc</td>
<td>Graphics context for drawing characters. The font selected into this GC must be the same as the <code>tkfont</code>.</td>
</tr>
<tr>
<td>int x, y</td>
<td>Coordinates at which to place the left edge of the baseline when displaying <code>string</code>.</td>
</tr>
<tr>
<td>int firstByte</td>
<td>The index of the first byte of the first character to underline in the <code>string</code>. Underlining begins at the left edge of this character.</td>
</tr>
<tr>
<td>int lastByte</td>
<td>The index of the first byte of the last character up to which the underline will be drawn. The character specified by <code>lastByte</code> will not itself be underlined.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

These routines are for measuring and displaying simple single-font, single-line strings. To measure and display single-font, multi-line,
justified text, refer to the documentation for `Tk_ComputeTextLayout`. There is no programming interface in the core of Tk that supports multi-font, multi-line text; support for that behavior must be built on top of simpler layers. Note that the interfaces described here are byte-oriented not character-oriented, so index values coming from Tcl scripts need to be converted to byte offsets using the `Tcl_UtfAtIndex` and related routines.

A glyph is the displayable picture of a letter, number, or some other symbol. Not all character codes in a given font have a glyph. Characters such as tabs, newlines/returns, and control characters that have no glyph are measured and displayed by these procedures in a platform-dependent manner; under X, they are replaced with backslashed escape sequences, while under Windows and Macintosh hollow or solid boxes may be substituted. Refer to the documentation for `Tk_ComputeTextLayout` for a programming interface that supports the platform-independent expansion of tab characters into columns and newlines/returns into multi-line text.

`Tk_MeasureChars` is used both to compute the length of a given string and to compute how many characters from a string fit in a given amount of space. The return value is the number of bytes from `string` that fit in the space specified by `maxPixels` subject to the conditions described by `flags`. If all characters fit, the return value will be `numBytes`. `*lengthPtr` is filled with the computed width, in pixels, of the portion of the string that was measured. For example, if the return value is 5, then `*lengthPtr` is filled with the distance between the left edge of `string[0]` and the right edge of `string[4]`.

`Tk_TextWidth` is a wrapper function that provides a simpler interface to the `Tk_MeasureChars` function. The return value is how much space in pixels the given `string` needs.

`Tk_DrawChars` draws the `string` at the given location in the given drawable.

`Tk_UnderlineChars` underlines the given range of characters in the given `string`. It does not draw the characters (which are assumed to
have been displayed previously by `Tk_DrawChars`; it just draws the underline. This procedure is used to underline a few characters without having to construct an underlined font. To produce natively underlined text, the appropriate underlined font should be constructed and used.

**SEE ALSO**

font, FontId

**KEYWORDS**

font, measurement

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1996 Sun Microsystems, Inc.
NAME
Tk_GetImage, Tk_RedrawImage, Tk_SizeOfImage, Tk_FreeImage - use an image in a widget

SYNOPSIS
#include <tk.h>
Tk_Image
Tk_GetImage(interp, tkwin, name, changeProc, clientData)
Tk_RedrawImage(image, imageX, imageY, width, height, drawable, drawableX, drawableY)
Tk_SizeOfImage(image, widthPtr, heightPtr)
Tk_FreeImage(image)

ARGUMENTS

DESCRIPTION

SEE ALSO

KEYWORDS
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp</em> (in)</td>
<td>Place to leave error message.</td>
</tr>
<tr>
<td>Tk_Window tkwin (in)</td>
<td>Window in which image will be used.</td>
</tr>
<tr>
<td>const char *name (in)</td>
<td>Name of image.</td>
</tr>
<tr>
<td>Tk_ImageChangedProc *changeProc (in)</td>
<td>Procedure for Tk to invoke whenever image content or size changes.</td>
</tr>
<tr>
<td>ClientData clientData (in)</td>
<td>One-word value for Tk to pass to <code>changeProc</code>.</td>
</tr>
<tr>
<td>Tk_Image image (in)</td>
<td>Token for image instance; must have been returned by a previous call to <code>Tk_GetImage</code>.</td>
</tr>
<tr>
<td>int imageX (in)</td>
<td>X-coordinate of upper-left corner of region of image to redisplay (measured in pixels from the image's upper-left corner).</td>
</tr>
<tr>
<td>int imageY (in)</td>
<td>Y-coordinate of upper-left corner of region of image to redisplay (measured in pixels from the image's upper-left corner).</td>
</tr>
<tr>
<td>int width ((in))</td>
<td>Width of region of image to redisplay.</td>
</tr>
<tr>
<td>int height ((in))</td>
<td>Height of region of image to redisplay.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Drawable <code>drawable</code> (in)</td>
<td>Where to display image. Must either be window specified to <code>Tk_GetImage</code> or a pixmap compatible with that window.</td>
</tr>
<tr>
<td>int <code>drawableX</code> (in)</td>
<td>Where to display image in <code>drawable</code>: this is the x-coordinate in <code>drawable</code> where x-coordinate <code>imageX</code> of the image should be displayed.</td>
</tr>
<tr>
<td>int <code>drawableY</code> (in)</td>
<td>Where to display image in <code>drawable</code>: this is the y-coordinate in <code>drawable</code> where y-coordinate <code>imageY</code> of the image should be displayed.</td>
</tr>
<tr>
<td>int <code>widthPtr</code> (out)</td>
<td>Store width of <code>image</code> (in pixels) here.</td>
</tr>
<tr>
<td>int <code>heightPtr</code> (out)</td>
<td>Store height of <code>image</code> (in pixels) here.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

These procedures are invoked by widgets that wish to display images. `Tk_GetImage` is invoked by a widget when it first decides to display an image. `name` gives the name of the desired image and `tkwin` identifies the window where the image will be displayed. `Tk_GetImage` looks up the image in the table of existing images and returns a token for a new instance of the image. If the image does not exist then `Tk_GetImage` returns NULL and leaves an error message in `interp->result`. 
When a widget wishes to actually display an image it must call 
**Tk_RedrawImage**, identifying the image (**image**), a region within the image to redisplay (**imageX**, **imageY**, **width**, and **height**), and a place to display the image (**drawable**, **drawableX**, and **drawableY**). Tk will then invoke the appropriate image manager, which will display the requested portion of the image before returning.

A widget can find out the dimensions of an image by calling 
**Tk_SizeOfImage**: the width and height will be stored in the locations given by **widthPtr** and **heightPtr**, respectively.

When a widget is finished with an image (e.g., the widget is being deleted or it is going to use a different image instead of the current one), it must call **Tk_FreeImage** to release the image instance. The widget should never again use the image token after passing it to **Tk_FreeImage**. There must be exactly one call to **Tk_FreeImage** for each call to **Tk_GetImage**.

If the contents or size of an image changes, then any widgets using the image will need to find out about the changes so that they can redisplay themselves. The **changeProc** and **clientData** arguments to 
**Tk_GetImage** are used for this purpose. **changeProc** will be called by Tk whenever a change occurs in the image; it must match the following prototype:

```c
typedef void Tk_ImageChangedProc(
    ClientData clientData,
    int x,
    int y,
    int width,
    int height,
    int imageWidth,
    int imageHeight);
```

The **clientData** argument to **changeProc** is the same as the **clientData** argument to **Tk_GetImage**. It is usually a pointer to the widget record
for the widget or some other data structure managed by the widget. The arguments $x$, $y$, \emph{width}, and \emph{height} identify a region within the image that must be redisplayed; they are specified in pixels measured from the upper-left corner of the image. The arguments \emph{imageWidth} and \emph{imageHeight} give the image's (new) size.

**SEE ALSO**

\texttt{Tk_CreateImageType}

**KEYWORDS**

\texttt{images, redisplay}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_DrawFocusHighlight - draw the traversal highlight ring for a widget

SYNOPSIS

#include <tk.h>
Tk_DrawFocusHighlight(tkwin, gc, width, drawable)

ARGUMENTS

<table>
<thead>
<tr>
<th>Tk_Window tkwin (in)</th>
<th>Window for which the highlight is being drawn. Used to retrieve the window's dimensions, among other things.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC gc (in)</td>
<td>Graphics context to use for drawing the highlight.</td>
</tr>
<tr>
<td>int width (in)</td>
<td>Width of the highlight ring, in pixels.</td>
</tr>
<tr>
<td>Drawable drawable (in)</td>
<td>Drawable in which to draw the highlight; usually an offscreen pixmap for double buffering.</td>
</tr>
</tbody>
</table>

DESCRIPTION

Tk_DrawFocusHighlight is a utility procedure that draws the traversal
highlight ring for a widget. It is typically invoked by widgets during redisplay.

**KEYWORDS**

focus, traversal highlight

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
**TkGetSelection** - retrieve the contents of a selection

### SYNOPSIS

```c
#include <tk.h>
int Tk_GetSelection(interp, tkwin, selection, target, proc, clientData)
```

### ARGUMENTS

<table>
<thead>
<tr>
<th><strong>Tcl_Interp</strong> <em>interp</em>* (in)</th>
<th>Interpreter to use for reporting errors.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tk_Window</strong> <em>tkwin</em>* (in)</td>
<td>Window on whose behalf to retrieve the selection (determines display from which to retrieve).</td>
</tr>
<tr>
<td><strong>Atom</strong> <em>selection</em>* (in)</td>
<td>The name of the selection to be retrieved.</td>
</tr>
<tr>
<td><strong>Atom</strong> <em>target</em>* (in)</td>
<td>Form in which to retrieve selection.</td>
</tr>
<tr>
<td><strong>Tk_GetSelProc</strong> <em>proc</em>* (in)</td>
<td>Procedure to invoke to process pieces of the selection as they are retrieved.</td>
</tr>
</tbody>
</table>
**DESCRIPTION**

**Tk_GetSelection** retrieves the selection specified by the atom *selection* in the format specified by *target*. The selection may actually be retrieved in several pieces; as each piece is retrieved, *proc* is called to process the piece. *Proc* should have arguments and result that match the type **Tk_GetSelProc**:

```c
typedef int Tk_GetSelProc(
    ClientData clientData,
    Tcl_Interp *interp,
    char *portion);
```

The *clientData* and *interp* parameters to *proc* will be copies of the corresponding arguments to **Tk_GetSelection**. *Portion* will be a pointer to a string containing part or all of the selection. For large selections, *proc* will be called several times with successive portions of the selection. The X Inter-Client Communication Conventions Manual allows a selection to be returned in formats other than strings, e.g. as an array of atoms or integers. If this happens, Tk converts the selection back into a string before calling *proc*. If a selection is returned as an array of atoms, Tk converts it to a string containing the atom names separated by white space. For any other format besides string, Tk converts a selection to a string containing hexadecimal values separated by white space.

**Tk_GetSelection** returns to its caller when the selection has been completely retrieved and processed by *proc*, or when a fatal error has occurred (e.g. the selection owner did not respond promptly).

**Tk_GetSelection** normally returns **TCL_OK**; if an error occurs, it returns **TCL_ERROR** and leaves an error message in *interp->result*. *Proc* should also return either **TCL_OK** or **TCL_ERROR**. If *proc*
encounters an error in dealing with the selection, it should leave an error message in \texttt{interp->result} and return \texttt{TCL\_ERROR}; this will abort the selection retrieval.

\begin{center}
\textbf{KEYWORDS}
\end{center}

\texttt{format, get, selection retrieval}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tk_AllocBitmapFromObj, Tk_GetBitmap,
Tk_GetBitmapFromObj, Tk_DefineBitmap, Tk_NameOfBitmap,
Tk_SizeOfBitmap, Tk_FreeBitmapFromObj, Tk_FreeBitmap
-maintain database of single-plane pixmaps

SYNOPSIS
#include <tk.h>
Pixmap
Tk_AllocBitmapFromObj(interp, tkwin, objPtr)
Pixmap
Tk_GetBitmap(interp, tkwin, info)
Pixmap
Tk_GetBitmapFromObj(tkwin, objPtr)
int
Tk_DefineBitmap(interp, name, source, width, height)
const char *
Tk_NameOfBitmap(display, bitmap)
Tk_SizeOfBitmap(display, bitmap, widthPtr, heightPtr)
Tk_FreeBitmapFromObj(tkwin, objPtr)
Tk_FreeBitmap(display, bitmap)

ARGUMENTS
DESCRIPTION
@fileName
name
error
gray75
gray50
gray25
gray12
hourglass
info
NAME

Tk_AllocBitmapFromObj, Tk_GetBitmap, Tk_GetBitmapFromObj,
Tk_DefineBitmap, Tk_NameOfBitmap, Tk_SizeOfBitmap,
Tk_FreeBitmapFromObj, Tk_FreeBitmap - maintain database of single-
plane pixmaps

SYNOPSIS

#include <tk.h>
Pixmap
Tk_AllocBitmapFromObj(interp, tkwin, objPtr)
Pixmap
Tk_GetBitmap(interp, tkwin, info)
Pixmap
Tk_GetBitmapFromObj(tkwin, objPtr)
int

Tk_DefineBitmap(*interp, name, source, width, height)
const char *

Tk_NameOfBitmap(*display, bitmap)

Tk_SizeOfBitmap(*display, bitmap, widthPtr, heightPtr)

Tk_FreeBitmapFromObj(tkwin, *objPtr)

Tk_FreeBitmap(*display, bitmap)

| ARGUMENTS |
|-----------------|--------------------------------------------------------------------------------------------------|
| **Tcl_Interp *interp (in)** | Interpreter to use for error reporting; if NULL then no error message is left after errors. |
| **Tk_Window tkwin (in)** | Token for window in which the bitmap will be used. |
| **Tcl_Obj *objPtr (in/out)** | String value describes desired bitmap; internal rep will be modified to cache pointer to corresponding Pixmap. |
| **const char *info (in)** | Same as *objPtr except description of bitmap is passed as a string and resulting Pixmap is not cached. |
| **const char *name (in)** | Name for new bitmap to be defined. |
| **const char *source (in)** | Data for bitmap, in standard bitmap format. Must be stored in static memory whose value will |
These procedures manage a collection of bitmaps (one-plane pixmaps) being used by an application. The procedures allow bitmaps to be re-used efficiently, thereby avoiding server overhead, and also allow bitmaps to be named with character strings.

**Tk_AllocBitmapFromObj** returns a Pixmap identifier for a bitmap that matches the description in *objPtr* and is suitable for use in *tkwin*. It re-uses an existing bitmap, if possible, and creates a new one otherwise. *ObjPtr*'s value must have one of the following forms:

- **@fileName**
  
  *FileName* must be the name of a file containing a bitmap description in the standard X11 or X10 format.

- **name**
Name must be the name of a bitmap defined previously with a call to Tk_DefineBitmap. The following names are pre-defined by Tk:

error
   The international “don't” symbol: a circle with a diagonal line across it.

gray75
   75% gray: a checkerboard pattern where three out of four bits are on.

gray50
   50% gray: a checkerboard pattern where every other bit is on.

gray25
   25% gray: a checkerboard pattern where one out of every four bits is on.

gray12
   12.5% gray: a pattern where one-eighth of the bits are on, consisting of every fourth pixel in every other row.

hourglass
   An hourglass symbol.

info
   A large letter “i”.

questhead
   The silhouette of a human head, with a question mark in it.

question
   A large question-mark.

warning
   A large exclamation point.

In addition, the following pre-defined names are available only on the Macintosh platform:
document
  A generic document.

stationery
  Document stationery.

edition
  The edition symbol.

application
  Generic application icon.

accessory
  A desk accessory.

folder
  Generic folder icon.

pfolder
  A locked folder.

trash
  A trash can.

floppy
  A floppy disk.

ramdisk
  A floppy disk with chip.

cdrom
  A cd disk icon.

preferences
  A folder with prefs symbol.

querydoc
  A database document icon.
stop
   A stop sign.

note
   A face with balloon words.

caution
   A triangle with an exclamation point.

Under normal conditions, \texttt{Tk\_AllocBitmapFromObj} returns an identifier for the requested bitmap. If an error occurs in creating the bitmap, such as when \texttt{objPtr} refers to a non-existent file, then \texttt{None} is returned and an error message is left in \texttt{interp}'s result if \texttt{interp} is not NULL. \texttt{Tk\_AllocBitmapFromObj} caches information about the return value in \texttt{objPtr}, which speeds up future calls to procedures such as \texttt{Tk\_AllocBitmapFromObj} and \texttt{Tk\_GetBitmapFromObj}.

\texttt{Tk\_GetBitmap} is identical to \texttt{Tk\_AllocBitmapFromObj} except that the description of the bitmap is specified with a string instead of an object. This prevents \texttt{Tk\_GetBitmap} from caching the return value, so \texttt{Tk\_GetBitmap} is less efficient than \texttt{Tk\_AllocBitmapFromObj}.

\texttt{Tk\_GetBitmapFromObj} returns the token for an existing bitmap, given the window and description used to create the bitmap. \texttt{Tk\_GetBitmapFromObj} does not actually create the bitmap; the bitmap must already have been created with a previous call to \texttt{Tk\_AllocBitmapFromObj} or \texttt{Tk\_GetBitmap}. The return value is cached in \texttt{objPtr}, which speeds up future calls to \texttt{Tk\_GetBitmapFromObj} with the same \texttt{objPtr} and \texttt{tkwin}.

\texttt{Tk\_DefineBitmap} associates a name with in-memory bitmap data so that the name can be used in later calls to \texttt{Tk\_AllocBitmapFromObj} or \texttt{Tk\_GetBitmap}. The \texttt{nameId} argument gives a name for the bitmap; it must not previously have been used in a call to \texttt{Tk\_DefineBitmap}. The arguments \texttt{source}, \texttt{width}, and \texttt{height} describe the bitmap. \texttt{Tk\_DefineBitmap} normally returns \texttt{TCL\_OK}; if an error occurs (e.g. a bitmap named \texttt{nameId} has already been defined) then \texttt{TCL\_ERROR} is returned and an error message is left in \texttt{interp->result}. Note:
Tk_DefineBitmap expects the memory pointed to by source to be static: Tk_DefineBitmap does not make a private copy of this memory, but uses the bytes pointed to by source later in calls to Tk_AllocBitmapFromObj or Tk_GetBitmap.

Typically Tk_DefineBitmap is used by #include-ing a bitmap file directly into a C program and then referencing the variables defined by the file. For example, suppose there exists a file stip.bitmap, which was created by the bitmap program and contains a stipple pattern. The following code uses Tk_DefineBitmap to define a new bitmap named foo:

```
Pixmap bitmap;
#include "stip.bitmap"
Tk_DefineBitmap(interp, "foo", stip_bits,
    stip_width, stip_height);
...
bitmap = Tk_GetBitmap(interp, tkwin, "foo");
```

This code causes the bitmap file to be read at compile-time and incorporates the bitmap information into the program's executable image. The same bitmap file could be read at run-time using Tk_GetBitmap:

```
Pixmap bitmap;
bitmap = Tk_GetBitmap(interp, tkwin, "@stip.bitmap")
```

The second form is a bit more flexible (the file could be modified after the program has been compiled, or a different string could be provided to read a different file), but it is a little slower and requires the bitmap file to exist separately from the program.

Tk maintains a database of all the bitmaps that are currently in use. Whenever possible, it will return an existing bitmap rather than creating
a new one. When a bitmap is no longer used, Tk will release it automatically. This approach can substantially reduce server overhead, so `Tk_AllocBitmapFromObj` and `Tk_GetBitmap` should generally be used in preference to Xlib procedures like `XReadBitmapFile`.

The bitmaps returned by `Tk_AllocBitmapFromObj` and `Tk_GetBitmap` are shared, so callers should never modify them. If a bitmap must be modified dynamically, then it should be created by calling Xlib procedures such as `XReadBitmapFile` or `XCreatePixmap` directly.

The procedure `Tk_NameOfBitmap` is roughly the inverse of `Tk_GetBitmap`. Given an X Pixmap argument, it returns the textual description that was passed to `Tk_GetBitmap` when the bitmap was created. `Bitmap` must have been the return value from a previous call to `Tk_AllocBitmapFromObj` or `Tk_GetBitmap`.

`Tk_SizeOfBitmap` returns the dimensions of its `bitmap` argument in the words pointed to by the `widthPtr` and `heightPtr` arguments. As with `Tk_NameOfBitmap`, `bitmap` must have been created by `Tk_AllocBitmapFromObj` or `Tk_GetBitmap`.

When a bitmap is no longer needed, `Tk_FreeBitmapFromObj` or `Tk_FreeBitmap` should be called to release it. For `Tk_FreeBitmapFromObj` the bitmap to release is specified with the same information used to create it; for `Tk_FreeBitmap` the bitmap to release is specified with its Pixmap token. There should be exactly one call to `Tk_FreeBitmapFromObj` or `Tk_FreeBitmap` for each call to `Tk_AllocBitmapFromObj` or `Tk_GetBitmap`.

**BUGS**

In determining whether an existing bitmap can be used to satisfy a new request, `Tk_AllocBitmapFromObj` and `Tk_GetBitmap` consider only the immediate value of the string description. For example, when a file name is passed to `Tk_GetBitmap`, `Tk_GetBitmap` will assume it is safe to re-use an existing bitmap created from the same file name: it will not check to see whether the file itself has changed, or whether the current directory has changed, thereby causing the name to refer to a different
file.

KEYWORDS

bitmap, pixmap

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1998 Sun Microsystems, Inc.
NAME

Tk_GetUid, Tk_Uid - convert from string to unique identifier

SYNOPSIS

#include <tk.h>
Tk_Uid
Tk_GetUid(string)

ARGUMENTS

char *string (in) String for which the corresponding unique identifier is desired.

DESCRIPTION

Tk_GetUid returns the unique identifier corresponding to string. Unique identifiers are similar to atoms in Lisp, and are used in Tk to speed up comparisons and searches. A unique identifier (type Tk_Uid) is a string pointer and may be used anywhere that a variable of type “char *” could be used. However, there is guaranteed to be exactly one unique identifier for any given string value. If Tk_GetUid is called twice, once with string a and once with string b, and if a and b have the same string value (strcmp(a, b) == 0), then Tk_GetUid will return exactly the same Tk_Uid value for each call (Tk_GetUid(a) == Tk_GetUid(b)). This means that variables of type Tk_Uid may be compared directly (x == y) without having to call strcmp. In addition, the return value from Tk_GetUid will have the same string value as its argument (strcmp(Tk_GetUid(a), a) == 0).
atom, unique identifier
NAME
Tk_AllocColorFromObj, Tk_GetColor, Tk_GetColorFromObj,
Tk_GetColorByValue, Tk_NameOfColor,
Tk_FreeColorFromObj, Tk_FreeColor - maintain database of
colors
SYNOPSIS
#include <tk.h>
XColor *
Tk_AllocColorFromObj(interp, tkwin, objPtr)
XColor *
Tk_GetColor(interp, tkwin, name)
XColor *
Tk_GetColorFromObj(tkwin, objPtr)
XColor *
Tk_GetColorByValue(tkwin, prefPtr)
const char *
Tk_NameOfColor(colorPtr)
GC
Tk_GCForColor(colorPtr, drawable)
Tk_FreeColorFromObj(tkwin, objPtr)
Tk_FreeColor(colorPtr)
ARGUMENTS
DESCRIPTION
colormame
#RGB
#RRGGBB
#RRRGGGBBB
#RRRGGGGBBB
KEYWORDS
# include <tk.h>

XColor *

**Tk_AllocColorFromObj**(*interp, tkwin, objPtr*)

XColor *

**Tk_GetColor**(*interp, tkwin, name*)

XColor *

**Tk_GetColorFromObj**(tkwin, objPtr)

XColor *

**Tk_GetColorByValue**(tkwin, prefPtr)

const char *

**Tk_NameOfColor**(colorPtr)

GC

**Tk_GCFForColor**(colorPtr, drawable)

**Tk_FreeColorFromObj**(tkwin, objPtr)

**Tk_FreeColor**(colorPtr)

## ARGUMENTS

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp</em> (in)</td>
<td>Interpreter to use for error reporting.</td>
</tr>
<tr>
<td>Tk_Window tkwin (in)</td>
<td>Token for window in which color will be used.</td>
</tr>
<tr>
<td>Tcl_Obj *objPtr (in/out)</td>
<td>String value describes desired color; internal rep will be modified to cache pointer to corresponding (XColor *).</td>
</tr>
<tr>
<td>char *name (in)</td>
<td>Same as objPtr except</td>
</tr>
</tbody>
</table>
DESCRIPTION

These procedures manage the colors being used by a Tk application. They allow colors to be shared whenever possible, so that colormap space is preserved, and they pick closest available colors when colormap space is exhausted.

Given a textual description of a color, Tk_AllocColorFromObj locates a pixel value that may be used to render the color in a particular window. The desired color is specified with an object whose string value
must have one of the following forms:

\textit{colorname}

Any of the valid textual names for a color defined in the server’s color database file, such as \texttt{red} or \texttt{PeachPuff}.

\texttt{#RGB}

\texttt{#RRGGBB}

\texttt{#RRRGGGGBBB}

\texttt{#RRRRGGGGGBBBB}

A numeric specification of the red, green, and blue intensities to use to display the color. Each \texttt{R}, \texttt{G}, or \texttt{B} represents a single hexadecimal digit. The four forms permit colors to be specified with 4-bit, 8-bit, 12-bit or 16-bit values. When fewer than 16 bits are provided for each color, they represent the most significant bits of the color. For example, \texttt{#3a7} is the same as \texttt{#3000a0007000}.

\texttt{Tk\_AllocColorFromObj} returns a pointer to an XColor structure; the structure indicates the exact intensities of the allocated color (which may differ slightly from those requested, depending on the limitations of the screen) and a pixel value that may be used to draw with the color in \texttt{tkwin}. If an error occurs in \texttt{Tk\_AllocColorFromObj} (such as an unknown color name) then NULL is returned and an error message is stored in \texttt{interp}'s result if \texttt{interp} is not NULL. If the colormap for \texttt{tkwin} is full, \texttt{Tk\_AllocColorFromObj} will use the closest existing color in the colormap. \texttt{Tk\_AllocColorFromObj} caches information about the return value in \texttt{objPtr}, which speeds up future calls to procedures such as \texttt{Tk\_AllocColorFromObj} and \texttt{Tk\_GetColorFromObj}.

\texttt{Tk\_GetColor} is identical to \texttt{Tk\_AllocColorFromObj} except that the description of the color is specified with a string instead of an object. This prevents \texttt{Tk\_GetColor} from caching the return value, so \texttt{Tk\_GetColor} is less efficient than \texttt{Tk\_AllocColorFromObj}.

\texttt{Tk\_GetColorFromObj} returns the token for an existing color, given the
window and description used to create the color. ***Tk_GetColorFromObj*** does not actually create the color; the color must already have been created with a previous call to ***Tk_AllocColorFromObj*** or ***Tk_GetColor***. The return value is cached in *objPtr*, which speeds up future calls to ***Tk_GetColorFromObj*** with the same *objPtr* and *tkwin*.

***Tk_GetColorByValue*** is similar to ***Tk_GetColor*** except that the desired color is indicated with the *red*, *green*, and *blue* fields of the structure pointed to by *colorPtr*.

This package maintains a database of all the colors currently in use. If the same color is requested multiple times from ***Tk_GetColor*** or ***Tk_AllocColorFromObj*** (e.g. by different windows), or if the same intensities are requested multiple times from ***Tk_GetColorByValue***, then existing pixel values will be re-used. Re-using an existing pixel avoids any interaction with the window server, which makes the allocation much more efficient. These procedures also provide a portable interface that works across all platforms. For this reason, you should generally use ***Tk_AllocColorFromObj***, ***Tk_GetColor***, or ***Tk_GetColorByValue*** instead of lower level procedures like **XAllocColor**.

Since different calls to this package may return the same shared pixel value, callers should never change the color of a pixel returned by the procedures. If you need to change a color value dynamically, you should use **XAllocColorCells** to allocate the pixel value for the color.

The procedure ***Tk_NameOfColor*** is roughly the inverse of ***Tk_GetColor***. If its *colorPtr* argument was created by ***Tk_AllocColorFromObj*** or ***Tk_GetColor*** then the return value is the string that was used to create the color. If *colorPtr* was created by a call to ***Tk_GetColorByValue***, or by any other mechanism, then the return value is a string that could be passed to ***Tk_GetColor*** to return the same color. Note: the string returned by ***Tk_NameOfColor*** is only guaranteed to persist until the next call to ***Tk_NameOfColor***.

***Tk_GCForColor*** returns a graphics context whose *foreground* field is the pixel allocated for *colorPtr* and whose other fields all have default
values. This provides an easy way to do basic drawing with a color. The graphics context is cached with the color and will exist only as long as colorPtr exists; it is freed when the last reference to colorPtr is freed by calling Tk_FreeColor.

When a color is no longer needed Tk_FreeColorFromObj or Tk_FreeColor should be called to release it. For Tk_FreeColorFromObj the color to release is specified with the same information used to create it; for Tk_FreeColor the color to release is specified with a pointer to its XColor structure. There should be exactly one call to Tk_FreeColorFromObj or Tk_FreeColor for each call to Tk_AllocColorFromObj, Tk_GetColor, or Tk_GetColorByValue.

**KEYWORDS**

color, intensity, object, pixel value

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1991 The Regents of the University of California.
Copyright © 1994-1998 Sun Microsystems, Inc.
NAME

Tk_GetUserInactiveTime, Tk_ResetUserInactiveTime - discover user inactivity time

SYNOPSIS

```
#include <tk.h>
long Tk_GetUserInactiveTime(display);
long Tk_ResetUserInactiveTime(display);
```

ARGUMENTS

Display *display (in)  The display on which the user inactivity timer is to be queried or reset.

DESCRIPTION

Tk_GetUserInactiveTime returns the number of milliseconds that have passed since the last user interaction (usually via keyboard or mouse) with the respective display. On systems and displays that do not support querying the user inactivity time, -1 is returned.

Tk_ResetUserInactiveTime resets the user inactivity timer of the given display to zero. On windowing systems that do not support multiple displays display can be passed as NULL.

KEYWORDS

idle, inactive
NAME
Tk_AllocCursorFromObj, Tk_GetCursor,
Tk_GetCursorFromObj, Tk_GetCursorFromData,
Tk_NameOfCursor, Tk_FreeCursorFromObj, Tk_FreeCursor -
maintain database of cursors

SYNOPSIS
#include <tk.h>
Tk_Cursor
TK_AllocCursorFromObj(interp, tkwin, objPtr)
Tk_Cursor
Tk_GetCursor(interp, tkwin, name)
Tk_Cursor
Tk_GetCursorFromObj(tkwin, objPtr)
Tk_Cursor
Tk_GetCursorFromData(interp, tkwin, source, mask, width,
height, xHot, yHot, fg, bg)
const char *
Tk_NameOfCursor(display, cursor)
Tk_FreeCursorFromObj(tkwin, objPtr)
Tk_FreeCursor(display, cursor)

ARGUMENTS

DESCRIPTION
name [fgColor [bgColor]]
@sourceName maskName fgColor bgColor
@sourceName fgColor
@sourceName

BUGS

KEYWORDS

NAME
Tk_AllocCursorFromObj, Tk_GetCursor, Tk_GetCursorFromObj, Tk_GetCursorFromData, Tk_NameOfCursor, Tk_FreeCursorFromObj, Tk_FreeCursor - maintain database of cursors

**SYNOPSIS**

```c
#include <tk.h>
Tk_Cursor
Tk_AllocCursorFromObj(interp, tkwin, objPtr)
Tk_Cursor
Tk_GetCursor(interp, tkwin, name)
Tk_Cursor
Tk_GetCursorFromObj(tkwin, objPtr)
Tk_Cursor
Tk_GetCursorFromData(interp, tkwin, source, mask, width, height, xHot, yHot, fg, bg)
const char *
Tk_NameOfCursor(display, cursor)
Tk_FreeCursorFromObj(tkwin, objPtr)
Tk_FreeCursor(display, cursor)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_Interp *interp</td>
<td>(in) Interpreter to use for error reporting.</td>
</tr>
<tr>
<td>Tk_Window tkwin</td>
<td>(in) Token for window in which the cursor will be used.</td>
</tr>
<tr>
<td>Tcl_Obj *objPtr</td>
<td>(in/out) Description of cursor; see below for possible values. Internal rep will be modified to cache pointer to corresponding Tk_Cursor.</td>
</tr>
<tr>
<td>char *name</td>
<td>(in) Same as objPtr except</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>const char *source (in)</td>
<td>Data for cursor cursor, in standard cursor format.</td>
</tr>
<tr>
<td>const char *mask (in)</td>
<td>Data for mask cursor, in standard cursor format.</td>
</tr>
<tr>
<td>int width (in)</td>
<td>Width of source and mask.</td>
</tr>
<tr>
<td>int height (in)</td>
<td>Height of source and mask.</td>
</tr>
<tr>
<td>int xHot (in)</td>
<td>X-location of cursor hot-spot.</td>
</tr>
<tr>
<td>int yHot (in)</td>
<td>Y-location of cursor hot-spot.</td>
</tr>
<tr>
<td>Tk_Uid fg (in)</td>
<td>Textual description of foreground color for cursor.</td>
</tr>
<tr>
<td>Tk_Uid bg (in)</td>
<td>Textual description of background color for cursor.</td>
</tr>
<tr>
<td>Display *display (in)</td>
<td>Display for which cursor was allocated.</td>
</tr>
<tr>
<td>Tk_Cursor cursor (in)</td>
<td>Opaque Tk identifier for cursor. If passed to Tk_FreeCursor, must have been returned by</td>
</tr>
</tbody>
</table>
DESCRIPTION

These procedures manage a collection of cursors being used by an application. The procedures allow cursors to be re-used efficiently, thereby avoiding server overhead, and also allow cursors to be named with character strings.

Tk_AllocCursorFromObj takes as argument an object describing a cursor, and returns an opaque Tk identifier for a cursor corresponding to the description. It re-uses an existing cursor if possible and creates a new one otherwise. Tk_AllocCursorFromObj caches information about the return value in objPtr, which speeds up future calls to procedures such as Tk_AllocCursorFromObj and Tk_GetCursorFromObj. If an error occurs in creating the cursor, such as when objPtr refers to a non-existent file, then None is returned and an error message will be stored in interp's result if interp is not NULL.

ObjPtr must contain a standard Tcl list with one of the following forms:

name [fgColor [bgColor]]

Name is the name of a cursor in the standard X cursor cursor, i.e., any of the names defined in cursorcursor.h, without the XC_. Some example values are X_cursor, hand2, or left_ptr. Appendix B of “The X Window System” by Scheifler & Gettys has illustrations showing what each of these cursors looks like. If fgColor and bgColor are both specified, they give the foreground and background colors to use for the cursor (any of the forms acceptable to Tk_GetColor may be used). If only fgColor is specified, then there will be no background color: the background will be transparent. If no colors are specified, then the cursor will use black for its foreground color and white for its background color.

The Macintosh version of Tk supports all of the X cursors and will
also accept any of the standard Mac cursors including **ibeam**, **crosshair**, **watch**, **plus**, and **arrow**. In addition, Tk will load Macintosh cursor resources of the types **crsr** (color) and **CURS** (black and white) by the name of the resource. The application and all its open dynamic library's resource files will be searched for the named cursor. If there are conflicts color cursors will always be loaded in preference to black and white cursors.

@**sourceName** maskName fgColor bgColor

In this form, **sourceName** and **maskName** are the names of files describing cursors for the cursor's source bits and mask. Each file must be in standard X11 or X10 cursor format. **FgColor** and **bgColor** indicate the colors to use for the cursor, in any of the forms acceptable to **Tk_GetColor**. This form of the command will not work on Macintosh or Windows computers.

@**sourceName** fgColor

This form is similar to the one above, except that the source is used as mask also. This means that the cursor's background is transparent. This form of the command will not work on Macintosh or Windows computers.

@**sourceName**

This form only works on Windows, and will load a Windows system cursor (.ani or .cur) from the file specified in **sourceName**.

**Tk_GetCursor** is identical to **Tk_AllocCursorFromObj** except that the description of the cursor is specified with a string instead of an object. This prevents **Tk_GetCursor** from caching the return value, so **Tk_GetCursor** is less efficient than **Tk_AllocCursorFromObj**.

**Tk_GetCursorFromObj** returns the token for an existing cursor, given the window and description used to create the cursor. **Tk_GetCursorFromObj** does not actually create the cursor; the cursor must already have been created with a previous call to **Tk_AllocCursorFromObj** or **Tk_GetCursor**. The return value is cached in **objPtr**, which speeds up future calls to **Tk_GetCursorFromObj** with the same **objPtr** and **tkwin**.
Tk_GetCursorFromData allows cursors to be created from in-memory descriptions of their source and mask cursors. Source points to standard cursor data for the cursor's source bits, and mask points to standard cursor data describing which pixels of source are to be drawn and which are to be considered transparent. Width and height give the dimensions of the cursor, xHot and yHot indicate the location of the cursor's hot-spot (the point that is reported when an event occurs), and fg and bg describe the cursor's foreground and background colors textually (any of the forms suitable for Tk_GetColor may be used). Typically, the arguments to Tk_GetCursorFromData are created by including a cursor file directly into the source code for a program, as in the following example:

```c
Tk_Cursor cursor;
#include "source.cursor"
#include "mask.cursor"
cursor = Tk_GetCursorFromData(interp, tkwin, source_bits, source_width, source_height, source_x
source_y_hot, Tk_GetUid("red"), Tk_GetUid("blue")
```

Under normal conditions Tk_GetCursorFromData will return an identifier for the requested cursor. If an error occurs in creating the cursor then None is returned and an error message will be stored in interp's result.

Tk_AllocCursorFromObj, Tk_GetCursor, and Tk_GetCursorFromData maintain a database of all the cursors they have created. Whenever possible, a call to Tk_AllocCursorFromObj, Tk_GetCursor, or Tk_GetCursorFromData will return an existing cursor rather than creating a new one. This approach can substantially reduce server overhead, so the Tk procedures should generally be used in preference to Xlib procedures like XCreateFontCursor or XCreatePixmapCursor, which create a new cursor on each call. The Tk procedures are also more portable than the lower-level X procedures.
The procedure `Tk_NameOfCursor` is roughly the inverse of `Tk_GetCursor`. If its cursor argument was created by `Tk_GetCursor`, then the return value is the name argument that was passed to `Tk_GetCursor` to create the cursor. If cursor was created by a call to `Tk_GetCursorFromData`, or by any other mechanism, then the return value is a hexadecimal string giving the X identifier for the cursor. Note: the string returned by `Tk_NameOfCursor` is only guaranteed to persist until the next call to `Tk_NameOfCursor`. Also, this call is not portable except for cursors returned by `Tk_GetCursor`.

When a cursor returned by `Tk_AllocCursorFromObj`, `Tk_GetCursor`, or `Tk_GetCursorFromData` is no longer needed, `Tk_FreeCursorFromObj` or `Tk_FreeCursor` should be called to release it. For `Tk_FreeCursorFromObj` the cursor to release is specified with the same information used to create it; for `Tk_FreeCursor` the cursor to release is specified with its `Tk_Cursor` token. There should be exactly one call to `Tk_FreeCursor` for each call to `Tk_AllocCursorFromObj`, `Tk_GetCursor`, or `Tk_GetCursorFromData`.

**BUGS**

In determining whether an existing cursor can be used to satisfy a new request, `Tk_AllocCursorFromObj`, `Tk_GetCursor`, and `Tk_GetCursorFromData` consider only the immediate values of their arguments. For example, when a file name is passed to `Tk_GetCursor`, `Tk_GetCursor` will assume it is safe to re-use an existing cursor created from the same file name: it will not check to see whether the file itself has changed, or whether the current directory has changed, thereby causing the name to refer to a different file. Similarly, `Tk_GetCursorFromData` assumes that if the same source pointer is used in two different calls, then the pointers refer to the same data; it does not check to see if the actual data values have changed.

**KEYWORDS**

cursor
NAME
Tk_GetVisual - translate from string to visual

SYNOPSIS
#include <tk.h>
Visual *
Tk_GetVisual(interp, tkwin, string, depthPtr, colormapPtr)

ARGUMENTS

class depth
default
pathName
number
best ?depth?
  (a)
  (b)
  (c)
  (d)

CREDITS

KEYWORDS

NAME
Tk_GetVisual - translate from string to visual

SYNOPSIS
#include <tk.h>
Visual *
Tk_GetVisual(interp, tkwin, string, depthPtr, colormapPtr)

ARGUMENTS

class depth
default
pathName
number
best ?depth?
  (a)
  (b)
  (c)
  (d)

CREDITS

KEYWORDS
### DESCRIPTION

**Tk_GetVisual** takes a string description of a visual and finds a suitable X Visual for use in *tkwin*, if there is one. It returns a pointer to the X Visual structure for the visual and stores the number of bits per pixel for it at *depthPtr*. If *string* is unrecognizable or if no suitable visual could be found, then NULL is returned and **Tk_GetVisual** leaves an error message in interp->result. If *colormap* is non-NULL then **Tk_GetVisual** also locates an appropriate colormap for use with the result visual and stores its X identifier at *colormapPtr*.

The *string* argument specifies the desired visual in one of the following ways:

- **class depth**
  The string consists of a class name followed by an integer depth, with any amount of white space (including none) in between. *class* selects what sort of visual is desired and must be one of

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_Interp <em>interp</em></em> (in)</td>
<td>Interpreter to use for error reporting.</td>
</tr>
<tr>
<td><strong>Tk_Window tkwin</strong> (in)</td>
<td>Token for window in which the visual will be used.</td>
</tr>
<tr>
<td>const char <em>string</em> (in)</td>
<td>String that identifies the desired visual. See below for valid formats.</td>
</tr>
<tr>
<td>int <em>depthPtr</em> (out)</td>
<td>Depth of returned visual gets stored here.</td>
</tr>
<tr>
<td>Colormap <em>colormapPtr</em> (out)</td>
<td>If non-NULL then a suitable colormap for visual is found and its identifier is stored here.</td>
</tr>
</tbody>
</table>
directcolor, grayscale, greyscale, pseudocolor, staticcolor, staticgray, staticgrey, or truecolor, or a unique abbreviation. depth specifies how many bits per pixel are needed for the visual. If possible, Tk_GetVisual will return a visual with this depth; if there is no visual of the desired depth then Tk_GetVisual looks first for a visual with greater depth, then one with less depth.

**default**
Use the default visual for tkwin's screen.

**pathName**
Use the visual for the window given by pathName. pathName must be the name of a window on the same screen as tkwin.

**number**
Use the visual whose X identifier is number.

**best ?depth?**
Choose the “best possible” visual, using the following rules, in decreasing order of priority:

(a)
a visual that has exactly the desired depth is best, followed by a visual with greater depth than requested (but as little extra as possible), followed by a visual with less depth than requested (but as great a depth as possible);

(b)
if no depth is specified, then the deepest available visual is chosen;

(c)
pseudocolor is better than truecolor or directcolor, which are better than staticcolor, which is better than staticgray or grayscale;

(d)
the default visual for the screen is better than any other visual.
CREDITS

The idea for **Tk_GetVisual**, and the first implementation, came from Paul Mackerras.

KEYWORDS

colormap, screen, visual

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
 Tk_ConfigureWindow, Tk_MoveWindow, Tk_ResizeWindow,
 Tk_MoveResizeWindow, Tk_SetWindowBorderWidth,
 Tk_ChangeWindowAttributes, Tk_SetWindowBackground,
 Tk_SetWindowBackgroundPixmap, Tk_SetWindowBorder,
 Tk_SetWindowBorderPixmap, Tk_SetWindowColormap,
 Tk_DefineCursor, Tk.UndefineCursor - change window
 configuration or attributes

SYNOPSIS
 #include <tk.h>
 Tk_ConfigureWindow(tkwin, valueMask, valuePtr)
 Tk_MoveWindow(tkwin, x, y)
 Tk_ResizeWindow(tkwin, width, height)
 Tk_MoveResizeWindow(tkwin, x, y, width, height)
 Tk_SetWindowBorderWidth(tkwin, borderWidth)
 Tk_ChangeWindowAttributes(tkwin, valueMask, attsPtr)
 Tk_SetWindowBackground(tkwin, pixel)
 Tk_SetWindowBackgroundPixmap(tkwin, pixmap)
 Tk_SetWindowBorder(tkwin, pixel)
 Tk_SetWindowBorderPixmap(tkwin, pixmap)
 Tk_SetWindowColormap(tkwin, colormap)
 Tk_DefineCursor(tkwin, cursor)
 Tk.UndefineCursor(tkwin)

ARGUMENTS
 DESCRIPTION
 BUGS
 SEE ALSO
 KEYWORDS

NAME
Tk_ConfigureWindow, Tk_MoveWindow, Tk_RESIZE_WINDOW,
Tk_MoveResizeWindow, Tk_SetWindowBorderWidth,
Tk_ChangeWindowAttributes, Tk_SetWindowBackground,
Tk_SetWindowBackgroundPixmap, Tk_SetWindowBorder,
Tk_SetWindowBorderPixmap, Tk_SetWindowColormap,
Tk_DefineCursor, Tk_UndefineCursor - change window configuration or
attributes

**SYNOPSIS**

```c
#include <tk.h>
Tk_ConfigureWindow(tkwin, valueMask, valuePtr)
Tk_MoveWindow(tkwin, x, y)
Tk_RESIZE_WINDOW(tkwin, width, height)
Tk_MoveResizeWindow(tkwin, x, y, width, height)
Tk_SetWindowBorderWidth(tkwin, borderWidth)
Tk_ChangeWindowAttributes(tkwin, valueMask, attsPtr)
Tk_SetWindowBackground(tkwin, pixel)
Tk_SetWindowBackgroundPixmap(tkwin, pixmap)
Tk_SetWindowBorder(tkwin, pixel)
Tk_SetWindowBorderPixmap(tkwin, pixmap)
Tk_SetWindowColormap(tkwin, colormap)
Tk_DefineCursor(tkwin, cursor)
Tk_UndefineCursor(tkwin)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Window tkwin (in)</td>
<td>Token for window.</td>
</tr>
<tr>
<td>unsigned int valueMask</td>
<td>OR-ed mask of values like CWX or CWBorderPixel, indicating which fields of</td>
</tr>
<tr>
<td></td>
<td>*valuePtr or *attsPtr to use.</td>
</tr>
<tr>
<td>XWindowChanges *valuePtr</td>
<td>Points to a structure containing new values for</td>
</tr>
</tbody>
</table>
the configuration parameters selected by *valueMask*. Fields not selected by *valueMask* are ignored.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int x (in)</td>
<td>New x-coordinate for <em>tkwin</em>'s top left pixel (including border, if any) within <em>tkwin</em>'s parent.</td>
</tr>
<tr>
<td>int y (in)</td>
<td>New y-coordinate for <em>tkwin</em>'s top left pixel (including border, if any) within <em>tkwin</em>'s parent.</td>
</tr>
<tr>
<td>int width (in)</td>
<td>New width for <em>tkwin</em> (interior, not including border).</td>
</tr>
<tr>
<td>int height (in)</td>
<td>New height for <em>tkwin</em> (interior, not including border).</td>
</tr>
<tr>
<td>int borderWidth (in)</td>
<td>New width for <em>tkwin</em>'s border.</td>
</tr>
<tr>
<td>XSetWindowAttributes *attsPtr (in)</td>
<td>Points to a structure containing new values for the attributes given by the <em>valueMask</em> argument. Attributes not selected by <em>valueMask</em> are ignored.</td>
</tr>
<tr>
<td>unsigned long pixel (in)</td>
<td>New background or border color for window.</td>
</tr>
</tbody>
</table>
**DESCRIPTION**

These procedures are analogous to the X library procedures with similar names, such as `XConfigureWindow`. Each one of the above procedures calls the corresponding X procedure and also saves the configuration information in Tk's local structure for the window. This allows the information to be retrieved quickly by the application (using macros such as `Tk_X` and `Tk_Height`) without having to contact the X server. In addition, if no X window has actually been created for `tkwin` yet, these procedures do not issue X operations or cause event handlers to be invoked; they save the information in Tk's local structure for the window; when the window is created later, the saved information will be used to configure the window.

See the X library documentation for details on what these procedures do and how they use their arguments.

In the procedures `Tk_ConfigureWindow`, `Tk_MoveWindow`, `Tk_ResizeWindow`, `Tk_MoveResizeWindow`, and `Tk_SetWindowBorderWidth`, if `tkwin` is an internal window then event
handlers interested in configure events are invoked immediately, before the procedure returns. If \textit{tkwin} is a top-level window then the event handlers will be invoked later, after \textit{X} has seen the request and returned an event for it.

Applications using Tk should never call procedures like \texttt{XConfigureWindow} directly; they should always use the corresponding Tk procedures.

The size and location of a window should only be modified by the appropriate geometry manager for that window and never by a window itself (but see \texttt{Tk\_MoveToplevelWindow} for moving a top-level window).

You may not use \texttt{Tk\_ConfigureWindow} to change the stacking order of a window (\textit{valueMask} may not contain the \texttt{CWSibling} or \texttt{CWStackMode} bits). To change the stacking order, use the procedure \texttt{Tk\_RestackWindow}.

The procedure \texttt{Tk\_SetWindowColormap} will automatically add \textit{tkwin} to the \texttt{TK\_COLORMAP\_WINDOWS} property of its nearest top-level ancestor if the new colormap is different from that of \textit{tkwin}'s parent and \textit{tkwin} is not already in the \texttt{TK\_COLORMAP\_WINDOWS} property.

\section*{BUGS}

\texttt{Tk\_SetWindowBackgroundPixmap} and \texttt{Tk\_SetWindowBorderPixmap} differ slightly from their Xlib counterparts in that the \textit{pixmap} argument may not necessarily be deleted immediately after calling one of these procedures. This is because \textit{tkwin}'s window may not exist yet at the time of the call, in which case \textit{pixmap} is merely saved and used later when \textit{tkwin}'s window is actually created. If you wish to delete \textit{pixmap}, then call \texttt{Tk\_MakeWindowExist} first to be sure that \textit{tkwin}'s window exists and \textit{pixmap} has been passed to the X server.

A similar problem occurs for the \textit{cursor} argument passed to \texttt{Tk\_DefineCursor}. The solution is the same as for pixmaps above: call
Tk_MakeWindowExist before freeing the cursor.

SEE ALSO

Tk_MoveToplevelWindow, Tk_RestackWindow

KEYWORDS

attributes, border, color, configure, height, pixel, pixmap, width, window, x, y
NAME
Tk_AllocFontFromObj, Tk_GetFont, Tk_GetFontFromObj, Tk_NameOfFont, Tk_FreeFontFromObj, Tk_FreeFont - maintain database of fonts

SYNOPSIS
#include <tk.h>
Tk_Font
Tk_AllocFontFromObj(interp, tkwin, objPtr)
Tk_Font
Tk_GetFont(interp, tkwin, string)
Tk_Font
Tk_GetFontFromObj(tkwin, objPtr)
const char *
Tk_NameOfFont(tkfont)
Tk_Font
Tk_FreeFontFromObj(tkwin, objPtr)
void
Tk_FreeFont(tkfont)

ARGUMENTS
DESCRIPTION
SEE ALSO
KEYWORDS

NAME
Tk_AllocFontFromObj, Tk_GetFont, Tk_GetFontFromObj, Tk_NameOfFont, Tk_FreeFontFromObj, Tk_FreeFont - maintain database of fonts

SYNOPSIS
#include <tk.h>
Tk_Font
Tk_AllocFontFromObj(interp, tkwin, objPtr)

Tk_Font
Tk_GetFont(interp, tkwin, string)

Tk_Font
Tk_GetFontFromObj(tkwin, objPtr)

const char *
Tk_NameOfFont(tkfont)

Tk_Font
Tk_FreeFontFromObj(tkwin, objPtr)

void
Tk_FreeFont(tkfont)

---

**ARGUMENTS**

<table>
<thead>
<tr>
<th>*<em>Tcl_Interp <em>interp (in)</em></em></th>
<th>Interpreter to use for error reporting. If <strong>NULL</strong>, then no error messages are left after errors.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tk_Window tkwin (in)</strong></td>
<td>Token for window in which font will be used.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>objPtr (in/out)</em></em></td>
<td>Gives name or description of font. See documentation for the <strong>font</strong> command for details on acceptable formats. Internal rep will be modified to cache corresponding Tk_Font.</td>
</tr>
<tr>
<td>*<em>const char <em>string (in)</em></em></td>
<td>Same as <strong>objPtr</strong> except description of font is passed as a string and resulting Tk_Font is not cached.</td>
</tr>
</tbody>
</table>
Tk_Font **tkfont** (in) Opaque font token.

**DESCRIPTION**

**Tk_AllocFontFromObj** finds the font indicated by **objPtr** and returns a token that represents the font. The return value can be used in subsequent calls to procedures such as **Tk_GetFontMetrics**, **Tk_MeasureChars**, and **Tk_FreeFont**. The Tk_Font token will remain valid until **Tk_FreeFontFromObj** or **Tk_FreeFont** is called to release it. **ObjPtr** can contain either a symbolic name or a font description; see the documentation for the **font** command for a description of the valid formats. If **Tk_AllocFontFromObj** is unsuccessful (because, for example, **objPtr** did not contain a valid font specification) then it returns **NULL** and leaves an error message in **interp**'s result if **interp** is not **NULL**. **Tk_AllocFontFromObj** caches information about the return value in **objPtr**, which speeds up future calls to procedures such as **Tk_AllocFontFromObj** and **Tk_GetFontFromObj**.

**Tk_GetFont** is identical to **Tk_AllocFontFromObj** except that the description of the font is specified with a string instead of an object. This prevents **Tk_GetFont** from caching the matching Tk_Font, so **Tk_GetFont** is less efficient than **Tk_AllocFontFromObj**.

**Tk_GetFontFromObj** returns the token for an existing font, given the window and description used to create the font. **Tk_GetFontFromObj** does not actually create the font; the font must already have been created with a previous call to **Tk_AllocFontFromObj** or **Tk_GetFont**. The return value is cached in **objPtr**, which speeds up future calls to **Tk_GetFontFromObj** with the same **objPtr** and **tkwin**.

**Tk_AllocFontFromObj** and **Tk_GetFont** maintain a database of all fonts they have allocated. If the same font is requested multiple times (e.g. by different windows or for different purposes), then a single Tk_Font will be shared for all uses. The underlying resources will be freed automatically when no-one is using the font anymore.

The procedure **Tk_NameOfFont** is roughly the inverse of **Tk_GetFont**.
Given a *tkfont* that was created by **Tk_GetFont** (or **Tk_AllocFontFromObj**), the return value is the *string* argument that was passed to **Tk_GetFont** to create the font. The string returned by **Tk_NameOfFont** is only guaranteed to persist until the *tkfont* is deleted. The caller must not modify this string.

When a font is no longer needed, **Tk_FreeFontFromObj** or **Tk_FreeFont** should be called to release it. For **Tk_FreeFontFromObj** the font to release is specified with the same information used to create it; for **Tk_FreeFont** the font to release is specified with its Tk_Font token. There should be exactly one call to **Tk_FreeFontFromObj** or **Tk_FreeFont** for each call to **Tk_AllocFontFromObj** or **Tk_GetFont**.

**SEE ALSO**

**Tk_FontId**

**KEYWORDS**

**font**

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1992 The Regents of the University of California.
Copyright © 1994-1998 Sun Microsystems, Inc.
NAME
Tk_FindPhoto, Tk_PhotoPutBlock, Tk_PhotoPutZoomedBlock, Tk_PhotoGetImage, Tk_PhotoBlank, Tk_PhotoExpand, Tk_PhotoGetSize, Tk_PhotoSetSize - manipulate the image data stored in a photo image.

SYNOPSIS
#include <tk.h>

Tk_PhotoHandle

Tk_FindPhoto(interp, imageName)

int

Tk_PhotoPutBlock(interp, handle, blockPtr, x, y, width, height, compRule)

int

Tk_PhotoPutZoomedBlock(interp, handle, blockPtr, x, y, width, height, zoomX, zoomY, subsampleX, subsampleY, compRule)

int

Tk_PhotoGetImage(handle, blockPtr)

void

Tk_PhotoBlank(handle)

int

Tk_PhotoExpand(interp, handle, width, height)

void

Tk_PhotoGetSize(handle, widthPtr, heightPtr)

int

Tk_PhotoSetSize(interp, handle, width, height)

ARGUMENTS
DESCRIPTION
PORTABILITY
CREDITS
KEYWORDS
NAME

Tk_FindPhoto, Tk_PhotoPutBlock, Tk_PhotoPutZoomedBlock, Tk_PhotoGetImage, Tk_PhotoBlank, Tk_PhotoExpand, Tk_PhotoGetSize, Tk_PhotoSetSize - manipulate the image data stored in a photo image.

SYNOPSIS

```c
#include <tk.h>
Tk_PhotoHandle
Tk_FindPhoto(interp, imageName)
int
Tk_PhotoPutBlock(interp, handle, blockPtr, x, y, width, height, compRule)
int
Tk_PhotoPutZoomedBlock(interp, handle, blockPtr, x, y, width, height, zoomX, zoomY, subsampleX, subsampleY, compRule)
int
Tk_PhotoGetImage(handle, blockPtr)
void
Tk_PhotoBlank(handle)
int
Tk_PhotoExpand(interp, handle, width, height)
void
Tk_PhotoGetSize(handle, widthPtr, heightPtr)
int
Tk_PhotoSetSize(interp, handle, width, height)
```

ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp</em> (in)</td>
<td>Interpreter in which image was created and in which error reporting is to be done.</td>
</tr>
<tr>
<td><strong>const char</strong> <em>imageName</em> (in)</td>
<td>Name of the photo image.</td>
</tr>
<tr>
<td><strong>Tk_PhotoHandle handle (in)</strong></td>
<td>Opaque handle identifying the photo image to be affected.</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>*<em>Tk_PhotolmageBlock <em>blockPtr (in)</em></em></td>
<td>Specifies the address and storage layout of image data.</td>
</tr>
<tr>
<td><strong>int x (in)</strong></td>
<td>Specifies the X coordinate where the top-left corner of the block is to be placed within the image.</td>
</tr>
<tr>
<td><strong>int y (in)</strong></td>
<td>Specifies the Y coordinate where the top-left corner of the block is to be placed within the image.</td>
</tr>
<tr>
<td><strong>int width (in)</strong></td>
<td>Specifies the width of the image area to be affected (for Tk_PhotoPutBlock) or the desired image width (for Tk_PhotoExpand and Tk_PhotoSetSize).</td>
</tr>
<tr>
<td><strong>int compRule (in)</strong></td>
<td>Specifies the compositing rule used when combining transparent pixels in a block of data with a photo image. Must be one of \begin{verbatim}TK_PHOTO_COMPOSITE\end{verbatim} (which puts the block of data over the top of the existing photo image, with the previous contents showing through in the transparent bits) or</td>
</tr>
</tbody>
</table>
TK_PHOTO_COMPOSITE_
(which discards the existing photo image contents in the rectangle covered by the data block.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int <strong>height</strong>  (in)</td>
<td>Specifies the height of the image area to be affected (for Tk_PhotoPutBlock) or the desired image height (for Tk_PhotoExpand and Tk_PhotoSetSize).</td>
</tr>
<tr>
<td>int <strong>widthPtr</strong> (out)</td>
<td>Pointer to location in which to store the image width.</td>
</tr>
<tr>
<td>int <strong>heightPtr</strong> (out)</td>
<td>Pointer to location in which to store the image height.</td>
</tr>
<tr>
<td>int <strong>subsampleX</strong> (in)</td>
<td>Specifies the subsampling factor in the X direction for input image data.</td>
</tr>
<tr>
<td>int <strong>subsampleY</strong> (in)</td>
<td>Specifies the subsampling factor in the Y direction for input image data.</td>
</tr>
<tr>
<td>int <strong>zoomX</strong> (in)</td>
<td>Specifies the zoom factor to be applied in the X direction to pixels being written to the photo image.</td>
</tr>
<tr>
<td>int <strong>zoomY</strong> (in)</td>
<td>Specifies the zoom factor to be applied in the Y direction to pixels being written to the photo image.</td>
</tr>
</tbody>
</table>
**DESCRIPTION**

**Tk_FindPhoto** returns an opaque handle that is used to identify a particular photo image to the other procedures. The parameter is the name of the image, that is, the name specified to the **image create photo** command, or assigned by that command if no name was specified.

**Tk_PhotoPutBlock** is used to supply blocks of image data to be displayed. The call affects an area of the image of size width x height pixels, with its top-left corner at coordinates (x,y). All of width, height, x, and y must be non-negative. If part of this area lies outside the current bounds of the image, the image will be expanded to include the area, unless the user has specified an explicit image size with the -width and/or -height widget configuration options (see photo(n)); in that case the area is silently clipped to the image boundaries.

The block parameter is a pointer to a **Tk_PhotoImageBlock** structure, defined as follows:

```c
typedef struct {
    unsigned char *pixelPtr;
    int width;
    int height;
    int pitch;
    int pixelSize;
    int offset[4];
} Tk_PhotoImageBlock;
```

The **pixelPtr** field points to the first pixel, that is, the top-left pixel in the block. The width and height fields specify the dimensions of the block of pixels. The pixelSize field specifies the address difference between two horizontally adjacent pixels. Often it is 3 or 4, but it can have any value. The pitch field specifies the address difference between two vertically adjacent pixels. The offset array contains the offsets from the address
of a pixel to the addresses of the bytes containing the red, green, blue and alpha (transparency) components. These are normally 0, 1, 2 and 3, but can have other values, e.g., for images that are stored as separate red, green and blue planes.

The `compRule` parameter to `Tk_PhotoPutBlock` specifies a compositing rule that says what to do with transparent pixels. The value `TK_PHOTO_COMPOSITE_OVERLAY` says that the previous contents of the photo image should show through, and the value `TK_PHOTO_COMPOSITE_SET` says that the previous contents of the photo image should be completely ignored, and the values from the block be copied directly across. The behavior in Tk8.3 and earlier was equivalent to having `TK_PHOTO_COMPOSITE_OVERLAY` as a compositing rule.

The value given for the `width` and `height` parameters to `Tk_PhotoPutBlock` do not have to correspond to the values specified in `block`. If they are smaller, `Tk_PhotoPutBlock` extracts a sub-block from the image data supplied. If they are larger, the data given are replicated (in a tiled fashion) to fill the specified area. These rules operate independently in the horizontal and vertical directions.

`Tk_PhotoPutBlock` normally returns `TCL_OK`, though if it cannot allocate sufficient memory to hold the resulting image, `TCL_ERROR` is returned instead and, if the `interp` argument is non-NULL, an error message is placed in the interpreter’s result.

`Tk_PhotoPutZoomedBlock` works like `Tk_PhotoPutBlock` except that the image can be reduced or enlarged for display. The `subsampleX` and `subsampleY` parameters allow the size of the image to be reduced by subsampling. `Tk_PhotoPutZoomedBlock` will use only pixels from the input image whose X coordinates are multiples of `subsampleX`, and whose Y coordinates are multiples of `subsampleY`. For example, an image of 512x512 pixels can be reduced to 256x256 by setting `subsampleX` and `subsampleY` to 2.

The `zoomX` and `zoomY` parameters allow the image to be enlarged by pixel replication. Each pixel of the (possibly subsampled) input image
will be written to a block \( zoomX \) pixels wide and \( zoomY \) pixels high of the displayed image. Subsampling and zooming can be used together for special effects.

\textbf{Tk\_PhotoGetImage} can be used to retrieve image data from a photo image. \textbf{Tk\_PhotoGetImage} fills in the structure pointed to by the \textit{blockPtr} parameter with values that describe the address and layout of the image data that the photo image has stored internally. The values are valid until the image is destroyed or its size is changed. \textbf{Tk\_PhotoGetImage} returns 1 for compatibility with the corresponding procedure in the old photo widget.

\textbf{Tk\_PhotoBlank} blanks the entire area of the photo image. Blank areas of a photo image are transparent.

\textbf{Tk\_PhotoExpand} requests that the widget's image be expanded to be at least \textit{width} \( \times \) \textit{height} pixels in size. The width and/or height are unchanged if the user has specified an explicit image width or height with the \texttt{-width} and/or \texttt{-height} configuration options, respectively. If the image data are being supplied in many small blocks, it is more efficient to use \textbf{Tk\_PhotoExpand} or \textbf{Tk\_PhotoSetSize} at the beginning rather than allowing the image to expand in many small increments as image blocks are supplied.

\textbf{Tk\_PhotoExpand} normally returns \texttt{TCL\_OK}, though if it cannot allocate sufficient memory to hold the resulting image, \texttt{TCL\_ERROR} is returned instead and, if the \textit{interp} argument is non-NULL, an error message is placed in the interpreter's result.

\textbf{Tk\_PhotoSetSize} specifies the size of the image, as if the user had specified the given \textit{width} and \textit{height} values to the \texttt{-width} and \texttt{-height} configuration options. A value of zero for \textit{width} or \textit{height} does not change the image's width or height, but allows the width or height to be changed by subsequent calls to \textbf{Tk\_PhotoPutBlock}, \textbf{Tk\_PhotoPutZoomedBlock} or \textbf{Tk\_PhotoExpand}.

\textbf{Tk\_PhotoSetSize} normally returns \texttt{TCL\_OK}, though if it cannot allocate sufficient memory to hold the resulting image, \texttt{TCL\_ERROR} is
returned instead and, if the *interp argument is non-NULL, an error message is placed in the interpreter's result.

**Tk_PhotoGetSize** returns the dimensions of the image in *widthPtr and *heightPtr.

### PORTABILITY

In Tk 8.3 and earlier, **Tk_PhotoPutBlock** and **Tk_PhotoPutZoomedBlock** had different signatures. If you want to compile code that uses the old interface against 8.4 without updating your code, compile it with the flag -DUSE_COMPOSITELESS_PHOTO_PUT_BLOCK. Code linked using Stubs against older versions of Tk will continue to work.

In Tk 8.4, **Tk_PhotoPutBlock**, **Tk_PhotoPutZoomedBlock**, **Tk_PhotoExpand** and **Tk_PhotoSetSize** did not take an *interp argument or return any result code. If insufficient memory was available for an image, Tk would panic. This behaviour is still supported if you compile your extension with the additional flag -DUSE_PANIC_ON_PHOTO_ALLOC_FAILURE. Code linked using Stubs against older versions of Tk will continue to work.

### CREDITS

The code for the photo image type was developed by Paul Mackerras, based on his earlier photo widget code.

### KEYWORDS

photo, image
NAME

Tk_GetVRootGeometry - Get location and size of virtual root for window

SYNOPSIS

#include <tk.h>
Tk_GetVRootGeometry(tkwin, xPtr, yPtr, widthPtr, heightPtr)

ARGUMENTS

Tk_Window tkwin (in)  Token for window whose virtual root is to be queried.

int xPtr (out)  Points to word in which to store x-offset of virtual root.

int yPtr (out)  Points to word in which to store y-offset of virtual root.

int widthPtr (out)  Points to word in which to store width of virtual root.

int heightPtr (out)  Points to word in which to store height of virtual root.

DESCRIPTION


**Tk_GetVRootGeometry** returns geometry information about the virtual root window associated with *tkwin*. The “associated” virtual root is the one in which *tkwin*'s nearest top-level ancestor (or *tkwin* itself if it is a top-level window) has been reparented by the window manager. This window is identified by a __SWM_ROOT or __WM_ROOT property placed on the top-level window by the window manager. If *tkwin* is not associated with a virtual root (e.g. because the window manager does not use virtual roots) then *xPtr* and *yPtr* will be set to 0 and *widthPtr* and *heightPtr* will be set to the dimensions of the screen containing *tkwin*.

**KEYWORDS**

geometry, height, location, virtual root, width, window manager

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
**NAME**

Tk_RestackWindow - Change a window's position in the stacking order

**SYNOPSIS**

```c
#include <tk.h>
int Tk_RestackWindow(tkwin, aboveBelow, other)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Window tkwin (in)</td>
<td>Token for window to restack.</td>
</tr>
<tr>
<td>int aboveBelow (in)</td>
<td>Indicates new position of tkwin relative to other; must be Above or Below.</td>
</tr>
<tr>
<td>Tk_Window other (in)</td>
<td>Tkwin will be repositioned just above or below this window. Must be a sibling of tkwin or a descendant of a sibling. If NULL then tkwin is restacked above or below all siblings.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

Tk_RestackWindow changes the stacking order of window relative to its siblings. If other is specified as NULL then window is repositioned at
the top or bottom of its stacking order, depending on whether \textit{aboveBelow} is \textbf{Above} or \textbf{Below}. If \textit{other} has a non-NULL value then \textit{window} is repositioned just above or below \textit{other}.

The \textit{aboveBelow} argument must have one of the symbolic values \textbf{Above} or \textbf{Below}. Both of these values are defined by the include file <X11/Xlib.h>.

\textbf{KEYWORDS}

\texttt{above}, \texttt{below}, \texttt{obscure}, \texttt{stacking\ order}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_GetHWND, Tk_AttachHWND - manage interactions between the Windows handle and an X window

SYNOPSIS

#include <tkPlatDecIs.h>
HWND
Tk_GetHWND(window)
Window
Tk_AttachHWND(tkwin, hwnd)

ARGUMENTS

Window window (in) X token for window.
Tk_Window tkwin (in) Tk window for window.
HWND hwnd (in) Windows HWND for window.

DESCRIPTION

Tk_GetHWND returns the Windows HWND identifier for X Windows window given by window.

Tk_AttachHWND binds the Windows HWND identifier to the specified Tk_Window given by tkwin. It returns an X Windows window that encapsulates the HWND.
KEYWORDS

identifier, window

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME
Tk_Grab, Tk_Ungrab - manipulate grab state in an application

SYNOPSIS
#include <tk.h>
int
Tk_Grab(interp, tkwin, grabGlobal)
void
Tk_Ungrab(tkwin)

ARGUMENTS

DESCRIPTION

KEYWORDS
grabGlobal

DESCRIPTION

These functions are used to set or release a global or application local grab. When a grab is set on a particular window in a Tk application, mouse and keyboard events can only be received by that window and its descendants. Mouse and keyboard events for windows outside the tree rooted at tkwin will be redirected to tkwin. If the grab is global, then all mouse and keyboard events for windows outside the tree rooted at tkwin (even those intended for windows in other applications) will be redirected to tkwin. If the grab is application local, only mouse and keyboard events intended for a window within the same application (but outside the tree rooted at tkwin) will be redirected.

Tk_Grab sets a grab on a particular window. Tkwin specifies the window on whose behalf the pointer is to be grabbed. GrabGlobal indicates whether the grab should be global or application local; if it is non-zero, it means the grab should be global. Normally, Tk_Grab returns TCL_OK; if an error occurs and the grab cannot be set, TCL_ERROR is returned and an error message is left if interp's result. Once this call completes successfully, no window outside the tree rooted at tkwin will receive pointer- or keyboard-related events until the next call to Tk_Ungrab. If a previous grab was in effect within the application, then it is replaced with a new one.

Tcl_Ungrab releases a grab on the mouse pointer and keyboard, if there is one set on the window given by tkwin. Once a grab is released, pointer and keyboard events will start being delivered to other windows again.

KEYWORDS
grab, window
NAME
Tk_CreateOptionTable, Tk_DeleteOptionTable, Tk_InitOptions, Tk_SetOptions, Tk_FreeSavedOptions, Tk_RestoreSavedOptions, Tk_GetOptionValue, Tk_GetOptionInfo, Tk_FreeConfigOptions, Tk_Offset - process configuration options

SYNOPSIS
#include <tk.h>
Tk_OptionTable
Tk_CreateOptionTable
 Tk_DeleteOptionTable
int
Tk_InitOptions
 Tk_InitOptions
int
Tk_SetOptions
 Tk_SetOptions
int
Tk_FreeSavedOptions
 Tk_FreeSavedOptions
Tcl_Obj *
Tk_GetOptionValue
 Tk_GetOptionValue
Tcl_Obj *
Tk_GetOptionInfo
 Tk_GetOptionInfo
Tcl_Obj *
Tk_FreeConfigOptions
 Tk_FreeConfigOptions
int
Tk_Offset
 Tk_Offset

ARGUMENTS
DESCRIPTION
TEMPLATES

TK_OPTION_ANCHOR
STORAGE MANAGEMENT ISSUES
OBJOFFSET VS. INTERNALOFFSET
CUSTOM OPTION TYPES
clientData
interp
Tkwin
valuePtr
recordPtr
internalOffset
saveInternalPtr
flags
KEYWORDS

NAME

Tk_CreateOptionTable, Tk_DeleteOptionTable, Tk_InitOptions, Tk_SetOptions, Tk_FreeSavedOptions, Tk_RestoreSavedOptions, Tk_GetOptionValue, Tk_GetOptionInfo, Tk_FreeConfigOptions, Tk_Offset - process configuration options
SYNOPSIS

#include <tk.h>
Tk_OptionTable
Tk_CreateOptionTable( interp, templatePtr )
Tk_DeleteOptionTable( optionTable )
int
Tk_InitOptions( interp, recordPtr, optionTable, tkwin )
int
Tk_SetOptions( interp, recordPtr, optionTable, objc, objv, tkwin, savePtr, maskPtr )
Tk_FreeSavedOptions( savedPtr )
Tk_RestoreSavedOptions( savedPtr )
Tcl_Obj *
Tk_GetOptionValue( interp, recordPtr, optionTable, namePtr, tkwin )
Tcl_Obj *
Tk_GetOptionInfo( interp, recordPtr, optionTable, namePtr, tkwin )
Tk_FreeConfigOptions( recordPtr, optionTable, tkwin )
int
Tk_Offset( type, field )

ARGUMENTS

Tcl_Interp *interp ( in )
A Tcl interpreter. Most procedures use this only for returning error messages; if it is NULL then no error messages are returned. For Tk_CreateOptionTable the value cannot be NULL; it gives the interpreter in which the option table will be used.

const Tk_OptionSpec *templatePtr ( in )
Points to an array of static information that describes
the configuration options that are supported. Used to build a Tk_OptionTable. The information pointed to by this argument must exist for the lifetime of the Tk_OptionTable.

<table>
<thead>
<tr>
<th>Tk_OptionTable optionTable (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Token for an option table. Must have been returned by a previous call to Tk_CreateOptionTable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>char *recordPtr (in/out)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Points to structure in which values of configuration options are stored; fields of this record are modified by procedures such as Tk_SetOptions and read by procedures such as Tk_GetOptionValue.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tk_Window tkwin (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>For options such as TK_OPTION_COLOR, this argument indicates the window in which the option will be used. If optionTable uses no window-dependent options, then a NULL value may be supplied for this argument.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>int objc (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of values in objv.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tcl_Obj *const objv[] (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command-line arguments for setting configuring options.</td>
</tr>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>Tk_SavedOptions *savePtr (out)</td>
</tr>
<tr>
<td>int *maskPtr (out)</td>
</tr>
<tr>
<td>Tk_SavedOptions *savedPtr (in/out)</td>
</tr>
<tr>
<td>Tcl_Obj *namePtr (in)</td>
</tr>
<tr>
<td>type name type (in)</td>
</tr>
<tr>
<td>field name field (in)</td>
</tr>
</tbody>
</table>
These procedures handle most of the details of parsing configuration options such as those for Tk widgets. Given a description of what options are supported, these procedures handle all the details of parsing options and storing their values into a C structure associated with the widget or object. The procedures were designed primarily for widgets in Tk, but they can also be used for other kinds of objects that have configuration options. In the rest of this manual page “widget” will be used to refer to the object whose options are being managed; in practice the object may not actually be a widget. The term “widget record” is used to refer to the C-level structure in which information about a particular widget or object is stored.

Note: the easiest way to learn how to use these procedures is to look at a working example. In Tk, the simplest example is the code that implements the button family of widgets, which is in tkButton.c. Other examples are in tkSquare.c and tkMenu.c.

In order to use these procedures, the code that implements the widget must contain a static array of Tk_OptionSpec structures. This is a template that describes the various options supported by that class of widget; there is a separate template for each kind of widget. The template contains information such as the name of each option, its type, its default value, and where the value of the option is stored in the widget record. See TEMPLATES below for more detail.

In order to process configuration options efficiently, the static template must be augmented with additional information that is available only at runtime. The procedure Tk_CreateOptionTable creates this dynamic information from the template and returns a Tk_OptionTable token that describes both the static and dynamic information. All of the other procedures, such as Tk_SetOptions, take a Tk_OptionTable token as argument. Typically, Tk_CreateOptionTable is called the first time that a widget of a particular class is created and the resulting Tk_OptionTable is used in the future for all widgets of that class. A Tk_OptionTable may be used only in a single interpreter, given by the
 interp argument to Tk_CreateOptionTable. When an option table is no longer needed Tk_DeleteOptionTable should be called to free all of its resources. All of the option tables for a Tcl interpreter are freed automatically if the interpreter is deleted.

Tk_InitOptions is invoked when a new widget is created to set the default values for all of the widget's configuration options. Tk_InitOptions is passed a token for an option table (optionTable) and a pointer to a widget record (recordPtr), which is the C structure that holds information about this widget. Tk_InitOptions uses the information in the option table to choose an appropriate default for each option, then it stores the default value directly into the widget record, overwriting any information that was already present in the widget record. Tk_InitOptions normally returns TCL_OK. If an error occurred while setting the default values (e.g., because a default value was erroneous) then TCL_ERROR is returned and an error message is left in interp's result if interp is not NULL.

Tk_SetOptions is invoked to modify configuration options based on information specified in a Tcl command. The command might be one that creates a new widget, or a command that modifies options on an existing widget. The objc and objv arguments describe the values of the arguments from the Tcl command. Objv must contain an even number of objects: the first object of each pair gives the name of an option and the second object gives the new value for that option. Tk_SetOptions looks up each name in optionTable, checks that the new value of the option conforms to the type in optionTable, and stores the value of the option into the widget record given by recordPtr. Tk_SetOptions normally returns TCL_OK. If an error occurred (such as an unknown option name or an illegal option value) then TCL_ERROR is returned and an error message is left in interp's result if interp is not NULL.

Tk_SetOptions has two additional features. First, if the maskPtr argument is not NULL then it points to an integer value that is filled in with information about the options that were modified. For each option in the template passed to Tk_CreateOptionTable there is a typeMask field. The bits of this field are defined by the code that implements the widget; for example, each bit might correspond to a particular
configuration option. Alternatively, bits might be used functionally. For example, one bit might be used for redisplay: all options that affect the widget's display, such that changing the option requires the widget to be redisplayed, might have that bit set. Another bit might indicate that the geometry of the widget must be recomputed, and so on.

**Tk_SetOptions** OR's together the *typeMask* fields from all the options that were modified and returns this value at *maskPtr*; the caller can then use this information to optimize itself so that, for example, it does not redisplay the widget if the modified options do not affect the widget's appearance.

The second additional feature of **Tk_SetOptions** has to do with error recovery. If an error occurs while processing configuration options, this feature makes it possible to restore all the configuration options to their previous values. Errors can occur either while processing options in **Tk_SetOptions** or later in the caller. In many cases the caller does additional processing after **Tk_SetOptions** returns; for example, it might use an option value to set a trace on a variable and may detect an error if the variable is an array instead of a scalar. Error recovery is enabled by passing in a non-NULL value for the savePtr argument to **Tk_SetOptions**; this should be a pointer to an uninitialized Tk_SavedOptions structure on the caller's stack. **Tk_SetOptions** overwrites the structure pointed to by savePtr with information about the old values of any options modified by the procedure. If **Tk_SetOptions** returns successfully, the caller uses the structure in one of two ways. If the caller completes its processing of the new options without any errors, then it must pass the structure to **Tk_FreeSavedOptions** so that the old values can be freed. If the caller detects an error in its processing of the new options, then it should pass the structure to **Tk_RestoreSavedOptions**, which will copy the old values back into the widget record and free the new values. If **Tk_SetOptions** detects an error then it automatically restores any options that had already been modified and leaves *savePtr in an empty state: the caller need not call either **Tk_FreeSavedOptions** or **Tk_RestoreSavedOptions**. If the savePtr argument to **Tk_SetOptions** is NULL then **Tk_SetOptions** frees each old option value immediately when it sets a new value for the option. In this case, if an error occurs in the third option, the old values
for the first two options cannot be restored.

Tk_GetOptionValue returns the current value of a configuration option for a particular widget. The namePtr argument contains the name of an option; Tk_GetOptionValue uses optionTable to lookup the option and extract its value from the widget record pointed to by recordPtr, then it returns an object containing that value. If an error occurs (e.g., because namePtr contains an unknown option name) then NULL is returned and an error message is left in interp's result unless interp is NULL.

Tk_GetOptionInfo returns information about configuration options in a form suitable for configure widget commands. If the namePtr argument is not NULL, it points to an object that gives the name of a configuration option; Tk_GetOptionInfo returns an object containing a list with five elements, which are the name of the option, the name and class used for the option in the option database, the default value for the option, and the current value for the option. If the namePtr argument is NULL, then Tk_GetOptionInfo returns information about all options in the form of a list of lists; each sublist describes one option. Synonym options are handled differently depending on whether namePtr is NULL: if namePtr is NULL then the sublist for each synonym option has only two elements, which are the name of the option and the name of the other option that it refers to; if namePtr is non-NULL and names a synonym option then the object returned is the five-element list for the other option that the synonym refers to. If an error occurs (e.g., because namePtr contains an unknown option name) then NULL is returned and an error message is left in interp's result unless interp is NULL.

Tk_FreeConfigOptions must be invoked when a widget is deleted. It frees all of the resources associated with any of the configuration options defined in recordPtr by optionTable.

The Tk_Offset macro is provided as a safe way of generating the objOffset and internalOffset values for entries in Tk_OptionSpec structures. It takes two arguments: the name of a type of record, and the name of a field in that record. It returns the byte offset of the named field in records of the given type.
The array of Tk_OptionSpec structures passed to
Tk_CreateOptionTable via its templatePtr argument describes the
configuration options supported by a particular class of widgets. Each
structure specifies one configuration option and has the following fields:

```c
typedef struct {
    Tk_OptionType type;
    const char *optionName;
    const char *dbName;
    const char *dbClass;
    const char *defValue;
    int objOffset;
    int internalOffset;
    int flags;
    ClientData clientData;
    int typeMask;
} Tk_OptionSpec;
```

The `type` field indicates what kind of configuration option this is (e.g.
TK_OPTION_COLOR for a color value, or TK_OPTION_INT for an
integer value). Type determines how the value of the option is parsed
(more on this below). The `optionName` field is a string such as -font or -bg;
it is the name used for the option in Tcl commands and passed to
procedures via the objc or namePtr arguments. The `dbName` and
`dbClass` fields are used by Tk_InitOptions to look up a default value for
this option in the option database; if `dbName` is NULL then the option
database is not used by Tk_InitOptions for this option. The `defValue`
field specifies a default value for this configuration option if no value is
specified in the option database. The `objOffset` and `internalOffset` fields
indicate where to store the value of this option in widget records (more
on this below); values for the `objOffset` and `internalOffset` fields should
always be generated with the Tk_Offset macro. The `flags` field contains
additional information to control the processing of this configuration
option (see below for details). `ClientData` provides additional type-
specific data needed by certain types. For instance, for
**TK_OPTION_COLOR** types, *clientData* is a string giving the default
value to use on monochrome displays. See the descriptions of the
different types below for details. The last field, *typeMask*, is used by
**Tk_SetOptions** to return information about which options were
modified; see the description of **Tk_SetOptions** above for details.

When **Tk_InitOptions** and **Tk_SetOptions** store the value of an option
into the widget record, they can do it in either of two ways. If the
*objOffset* field of the Tk_OptionSpec is greater than or equal to zero,
then the value of the option is stored as a (Tcl_Obj *) at the location in
the widget record given by *objOffset*. If the *internalOffset* field of the
Tk_OptionSpec is greater than or equal to zero, then the value of the
option is stored in a type-specific internal form at the location in the
widget record given by *internalOffset*. For example, if the option's type
is **TK_OPTION_INT** then the internal form is an integer. If the *objOffset*
or *internalOffset* field is negative then the value is not stored in that
form. At least one of the offsets must be greater than or equal to zero.

The *flags* field consists of one or more bits ORed together. At present
only a single flag is supported: **TK_OPTION_NULL_OK**. If this bit is set
for an option then an empty string will be accepted as the value for the
option and the resulting internal form will be a NULL pointer, a zero
value, or **None**, depending on the type of the option. If the flag is not set
then empty strings will result in errors. **TK_OPTION_NULL_OK** is
typically used to allow a feature to be turned off entirely, e.g. set a
cursor value to **None** so that a window simply inherits its parent's
cursor. Not all option types support the **TK_OPTION_NULL_OK** flag;
for those that do, there is an explicit indication of that fact in the
descriptions below.

The *type* field of each Tk_OptionSpec structure determines how to
parse the value of that configuration option. The legal value for *type*,
and the corresponding actions, are described below. If the type requires
a *tkwin* value to be passed into procedures like **Tk_SetOptions**, or if it
uses the *clientData* field of the Tk_OptionSpec, then it is indicated
explicitly; if not mentioned, the type requires neither *tkwin* nor
*clientData*. 
**TK_OPTION_ANCHOR**
The value must be a standard anchor position such as **ne** or **center**. The internal form is a Tk_Anchor value like the ones returned by **Tk_GetAnchorFromObj**.

**TK_OPTION_BITMAP**
The value must be a standard Tk bitmap name. The internal form is a Pixmap token like the ones returned by **Tk_AllocBitmapFromObj**. This option type requires *tkwin* to be supplied to procedures such as **Tk_SetOptions**, and it supports the **TK_OPTION_NULL_OK** flag.

**TK_OPTION_BOOLEAN**
The value must be a standard boolean value such as **true** or **no**. The internal form is an integer with value 0 or 1.

**TK_OPTION_BORDER**
The value must be a standard color name such as **red** or **#ff8080**. The internal form is a Tk_3DBorder token like the ones returned by **Tk_Alloc3DBorderFromObj**. This option type requires *tkwin* to be supplied to procedures such as **Tk_SetOptions**, and it supports the **TK_OPTION_NULL_OK** flag.

**TK_OPTION_COLOR**
The value must be a standard color name such as **red** or **#ff8080**. The internal form is an (XColor *) token like the ones returned by **Tk_AllocColorFromObj**. This option type requires *tkwin* to be supplied to procedures such as **Tk_SetOptions**, and it supports the **TK_OPTION_NULL_OK** flag.

**TK_OPTION_CURSOR**
The value must be a standard cursor name such as **cross** or **@foo**. The internal form is a Tk_Cursor token like the ones returned by **Tk_AllocCursorFromObj**. This option type requires *tkwin* to be supplied to procedures such as **Tk_SetOptions**, and when the option is set the cursor for the window is changed by calling **XDefineCursor**. This option type also supports the
TK_OPTION_NULL_OK flag.

TK_OPTION_CUSTOM
This option allows applications to define new option types. The clientData field of the entry points to a structure defining the new option type. See the section CUSTOM OPTION TYPES below for details.

TK_OPTION_DOUBLE
The string value must be a floating-point number in the format accepted by strtol. The internal form is a C double value. This option type supports the TK_OPTION_NULL_OK flag; if a NULL value is set, the internal representation is set to zero.

TK_OPTION_END
Marks the end of the template. There must be a Tk_OptionSpec structure with type TK_OPTION_END at the end of each template. If the clientData field of this structure is not NULL, then it points to an additional array of Tk_OptionSpec's, which is itself terminated by another TK_OPTION_END entry. Templates may be chained arbitrarily deeply. This feature allows common options to be shared by several widget classes.

TK_OPTION_FONT
The value must be a standard font name such as Times 16. The internal form is a Tk_Font handle like the ones returned by Tk_AllocFontFromObj. This option type requires tkwin to be supplied to procedures such as Tk_SetOptions, and it supports the TK_OPTION_NULL_OK flag.

TK_OPTION_INT
The string value must be an integer in the format accepted by strtol (e.g. 0 and 0x prefixes may be used to specify octal or hexadecimal numbers, respectively). The internal form is a C int value.

TK_OPTION_JUSTIFY
The value must be a standard justification value such as left. The
internal form is a Tk_Justify like the values returned by
Tk_GetJustifyFromObj.

**TK_OPTION_PIXELS**
The value must specify a screen distance such as \(2i\) or \(6.4\). The
internal form is an integer value giving a distance in pixels, like the
values returned by Tk_GetPixelsFromObj. Note: if the objOffset
field is not used then information about the original value of this
option will be lost. See OBJOFFSET VS. INTERNALOFFSET
below for details. This option type supports the
**TK_OPTION_NULL_OK** flag; if a NULL value is set, the internal
representation is set to zero.

**TK_OPTION_RELIEF**
The value must be standard relief such as **raised**. The internal form
is an integer relief value such as **TK_RELIEF_RAISED**. This option
type supports the **TK_OPTION_NULL_OK** flag; if the empty string
is specified as the value for the option, the integer relief value is set
to **TK_RELIEF_NULL**.

**TK_OPTION_STRING**
The value may be any string. The internal form is a (char *) pointer
that points to a dynamically allocated copy of the value. This option
type supports the **TK_OPTION_NULL_OK** flag.

**TK_OPTION_STRING_TABLE**
For this type, clientData is a pointer to an array of strings suitable
for passing to Tcl_GetIndexFromObj. The value must be one of
the strings in the table, or a unique abbreviation of one of the
strings. The internal form is an integer giving the index into the
table of the matching string, like the return value from
Tcl_GetStringFromObj.

**TK_OPTION_SYNONYM**
This type is used to provide alternative names for an option (for
example, -bg is often used as a synonym for -background). The
clientData field is a (char *) pointer that gives the name of another
option in the same table. Whenever the synonym option is used,
the information from the other option will be used instead.

**TK_OPTION_WINDOW**
The value must be a window path name. The internal form is a Tk_Window token for the window. This option type requires *tkwin* to be supplied to procedures such as Tk_SetOptions (in order to identify the application), and it supports the **TK_OPTION_NULL_OK** flag.

---

**STORAGE MANAGEMENT ISSUES**

If a field of a widget record has its offset stored in the *objOffset* or *internalOffset* field of a Tk_OptionSpec structure then the procedures described here will handle all of the storage allocation and resource management issues associated with the field. When the value of an option is changed, Tk_SetOptions (or Tk_FreeSavedOptions) will automatically free any resources associated with the old value, such as Tk_Fonts for **TK_OPTION_FONT** options or dynamically allocated memory for **TK_OPTION_STRING** options. For an option stored as an object using the *objOffset* field of a Tk_OptionSpec, the widget record shares the object pointed to by the *objv* value from the call to Tk_SetOptions. The reference count for this object is incremented when a pointer to it is stored in the widget record and decremented when the option is modified. When the widget is deleted Tk_FreeConfigOptions should be invoked; it will free the resources associated with all options and decrement reference counts for any objects.

However, the widget code is responsible for storing NULL or None in all pointer and token fields before invoking Tk_InitOptions. This is needed to allow proper cleanup in the rare case where an error occurs in Tk_InitOptions.

---

**OBJOFFSET VS. INTERNALOFFSET**

In most cases it is simplest to use the *internalOffset* field of a Tk_OptionSpec structure and not the *objOffset* field. This makes the internal form of the value immediately available to the widget code so
the value does not have to be extracted from an object each time it is used. However, there are two cases where the objOffset field is useful. The first case is for TK_OPTION_PIXELS options. In this case, the internal form is an integer pixel value that is valid only for a particular screen. If the value of the option is retrieved, it will be returned as a simple number. For example, after the command .b configure -borderwidth 2m, the command .b configure -borderwidth might return 7, which is the integer pixel value corresponding to 2m. Unfortunately, this loses the original screen-independent value. Thus for TK_OPTION_PIXELS options it is better to use the objOffset field. In this case the original value of the option is retained in the object and can be returned when the option is retrieved. In most cases it is convenient to use the internalOffset field as well, so that the integer value is immediately available for use in the widget code (alternatively, Tk_GetPixelsFromObj can be used to extract the integer value from the object whenever it is needed). Note: the problem of losing information on retrievals exists only for TK_OPTION_PIXELS options.

The second reason to use the objOffset field is in order to implement new types of options not supported by these procedures. To implement a new type of option, you can use TK_OPTION_STRING as the type in the Tk_OptionSpec structure and set the objOffset field but not the internalOffset field. Then, after calling Tk_SetOptions, convert the object to internal form yourself.

### CUSTOM OPTION TYPES

Applications can extend the built-in configuration types with additional configuration types by writing procedures to parse, print, free, and restore saved copies of the type and creating a structure pointing to those procedures:

```c
typedef struct Tk_ObjCustomOption {
    char *name;
    Tk_CustomOptionSetProc *setProc;
    Tk_CustomOptionGetProc *getProc;
    Tk_CustomOptionRestoreProc *restoreProc;
} Tk_ObjCustomOption;
```
The Tk_ObjCustomOption structure contains six fields: a name for the custom option type; pointers to the four procedures; and a `clientData` value to be passed to those procedures when they are invoked. The `clientData` value typically points to a structure containing information that is needed by the procedures when they are parsing and printing.
options. RestoreProc and freeProc may be NULL, indicating that no function should be called for those operations.

The setProc procedure is invoked by Tk_SetOptions to convert a Tcl_Obj into an internal representation and store the resulting value in the widget record. The arguments are:

clientData
A copy of the clientData field in the Tk_ObjCustomOption structure.

interp
A pointer to a Tcl interpreter, used for error reporting.

Tkwin
A copy of the tkwin argument to Tk_SetOptions

valuePtr
A pointer to a reference to a Tcl_Obj describing the new value for the option; it could have been specified explicitly in the call to Tk_SetOptions or it could come from the option database or a default. If the objOffset for the option is non-negative (the option value is stored as a (Tcl_Obj *) in the widget record), the Tcl_Obj pointer referenced by valuePtr is the pointer that will be stored at the objOffset for the option. SetProc may modify the value if necessary; for example, setProc may change the value to NULL to support the TK_OPTION_NULL_OK flag.

recordPtr
A pointer to the start of the widget record to modify.

internalOffset
Offset in bytes from the start of the widget record to the location where the internal representation of the option value is to be placed.

saveInternalPtr
A pointer to storage allocated in a Tk_SavedOptions structure for the internal representation of the original option value. Before
setting the option to its new value, setProc should set the value referenced by saveInternalPtr to the original value of the option in order to support Tk_RestoreSavedOptions.

flags
A copy of the flags field in the Tk_OptionSpec structure for the option

SetProc returns a standard Tcl result: TCL_OK to indicate successful processing, or TCL_ERROR to indicate a failure of any kind. An error message may be left in the Tcl interpreter given by interp in the case of an error.

The getProc procedure is invoked by Tk_GetOptionValue and Tk_GetOptionInfo to retrieve a Tcl_Obj representation of the internal representation of an option. The clientData argument is a copy of the clientData field in the Tk_ObjCustomOption structure. Tkwin is a copy of the tkwin argument to Tk_GetOptionValue or Tk_GetOptionInfo. RecordPtr is a pointer to the beginning of the widget record to query. InternalOffset is the offset in bytes from the beginning of the widget record to the location where the internal representation of the option value is stored. GetProc must return a pointer to a Tcl_Obj representing the value of the option.

The restoreProc procedure is invoked by Tk_RestoreSavedOptions to restore a previously saved internal representation of a custom option value. The clientData argument is a copy of the clientData field in the Tk_ObjCustomOption structure. Tkwin is a copy of the tkwin argument to Tk_GetOptionValue or Tk_GetOptionInfo. InternalPtr is a pointer to the location where internal representation of the option value is stored. SaveInternalPtr is a pointer to the saved value. RestoreProc must copy the value from saveInternalPtr to internalPtr to restore the value. RestoreProc need not free any memory associated with either internalPtr or saveInternalPtr; freeProc will be invoked to free that memory if necessary. RestoreProc has no return value.

The freeProc procedure is invoked by Tk_SetOptions and Tk_FreeSavedOptions to free any storage allocated for the internal
representation of a custom option. The clientData argument is a copy of the clientData field in the Tk_ObjCustomOption structure. Tkwin is a copy of the tkwin argument to Tk_GetOptionValue or Tk_GetOptionInfo. InternalPtr is a pointer to the location where the internal representation of the option value is stored. The freeProc must free any storage associated with the option. FreeProc has no return value.

<table>
<thead>
<tr>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>anchor, bitmap, boolean, border, color, configuration option, cursor, double, font, integer, justify, pixels, relief, screen distance, synonym</td>
</tr>
</tbody>
</table>
NAME

Tk_HandleEvent - invoke event handlers for window system events

SYNOPSIS

```
#include <tk.h>
Tk_HandleEvent(eventPtr)
```

ARGUMENTS

```
XEvent *eventPtr (in) Pointer to X event to dispatch to relevant handler(s).
```

DESCRIPTION

**Tk_HandleEvent** is a lower-level procedure that deals with window events. It is called by **Tcl_ServiceEvent** (and indirectly by **Tk_DoOneEvent**), and in a few other cases within Tk. It makes callbacks to any window event handlers (created by calls to **Tk_CreateEventHandler**) that match `eventPtr` and then returns. In some cases it may be useful for an application to bypass the Tk event queue and call **Tk_HandleEvent** directly instead of calling **Tcl_QueueEvent** followed by **Tcl_ServiceEvent**.

This procedure may be invoked recursively. For example, it is possible to invoke **Tk_HandleEvent** recursively from a handler called by **Tk_HandleEvent**. This sort of operation is useful in some modal situations, such as when a notifier has been popped up and an application wishes to wait for the user to click a button in the notifier.
before doing anything else.

**KEYWORDS**

callback, event, handler, window

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1992 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_RestrictEvents - filter and selectively delay X events

SYNOPSIS

#include <tk.h>
Tk_RestrictProc *
Tk_RestrictEvents(proc, clientData, prevClientDataPtr)

ARGUMENTS

Tk_RestrictProc *proc (in)  
Predicate procedure to call to filter incoming X events. NULL means do not restrict events at all.

ClientData clientData (in)  
Arbitrary argument to pass to proc.

ClientData *prevClientDataPtr (out)  
Pointer to place to save argument to previous restrict procedure.

DESCRIPTION

This procedure is useful in certain situations where applications are only prepared to receive certain X events. After Tk_RestrictEvents is called, Tk_DoOneEvent (and hence Tk_MainLoop) will filter X input events through proc. Proc indicates whether a given event is to be processed immediately, deferred until some later time (e.g. when the
event restriction is lifted), or discarded. *Proc* is a procedure with arguments and result that match the type **Tk_RestrictProc**:

```c
typedef Tk_RestrictAction Tk_RestrictProc(
    ClientData clientData, 
    XEvent *eventPtr);
```

The *clientData* argument is a copy of the *clientData* passed to **Tk_RestrictEvents**; it may be used to provide proc with information it needs to filter events. The *eventPtr* points to an event under consideration. *Proc* returns a restrict action (enumerated type **Tk_RestrictAction**) that indicates what **Tk_DoOneEvent** should do with the event. If the return value is **TK_PROCESS_EVENT**, then the event will be handled immediately. If the return value is **TK_DEFER_EVENT**, then the event will be left on the event queue for later processing. If the return value is **TK_DISCARD_EVENT**, then the event will be removed from the event queue and discarded without being processed.

**Tk_RestrictEvents** uses its return value and *prevClientDataPtr* to return information about the current event restriction procedure (a NULL return value means there are currently no restrictions). These values may be used to restore the previous restriction state when there is no longer any need for the current restriction.

There are very few places where **Tk_RestrictEvents** is needed. In most cases, the best way to restrict events is by changing the bindings with the **bind** Tcl command or by calling **Tk_CreateEventHandler** and **Tk_DeleteEventHandler** from C. The main place where **Tk_RestrictEvents** must be used is when performing synchronous actions (for example, if you need to wait for a particular event to occur on a particular window but you do not want to invoke any handlers for any other events). The “obvious” solution in these situations is to call **XNextEvent** or **XWindowEvent**, but these procedures cannot be used because Tk keeps its own event queue that is separate from the X event queue. Instead, call **Tk_RestrictEvents** to set up a filter, then call
Tk_DoOneEvent to retrieve the desired event(s).

**KEYWORDS**

delay, event, filter, restriction

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tk_CreateBindingTable, Tk_DeleteBindingTable,
Tk_CreateBinding, Tk_DeleteBinding, Tk_GetBinding,
Tk_GetAllBindings, Tk_DeleteAllBindings, Tk_BindEvent -
invoke scripts in response to X events

SYNOPSIS
#include <tk.h>
Tk_BindingTable
Tk_CreateBindingTable(interp)
Tk_DeleteBindingTable(bindingTable)
unsigned long
Tk_CreateBinding(interp, bindingTable, object, eventString,
script, append)
int
Tk_DeleteBinding(interp, bindingTable, object, eventString)
const char *
Tk_GetBinding(interp, bindingTable, object, eventString)
Tk_GetAllBindings(interp, bindingTable, object)
Tk_DeleteAllBindings(bindingTable, object)
Tk_BindEvent(bindingTable, eventPtr, tkwin, numObjects,
objectPtr)

ARGUMENTS
DESCRIPTION
KEYWORDS

NAME
Tk_CreateBindingTable, Tk_DeleteBindingTable, Tk_CreateBinding,
Tk_DeleteBinding, Tk_GetBinding, Tk_GetAllBindings,
Tk_DeleteAllBindings, Tk_BindEvent - invoke scripts in response to X events
#include <tk.h>

**Synopsis**

```c
Tk_BindingTable
Tk_CreateBindingTable(interp)
Tk_DeleteBindingTable(bindingTable)
unsigned long
Tk_CreateBinding(interp, bindingTable, object, eventString, script, append)
int
Tk_DeleteBinding(interp, bindingTable, object, eventString)
const char *
Tk_GetBinding(interp, bindingTable, object, eventString)
Tk_GetAllBindings(interp, bindingTable, object)
Tk_DeleteAllBindings(bindingTable, object)
Tk_BindEvent(bindingTable, eventPtr, tkwin, numObjects, objectPtr)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_Interp <em>interp</em></em> (in)</td>
<td>Interpreter to use when invoking bindings in binding table. Also used for returning results and errors from binding procedures.</td>
</tr>
<tr>
<td><strong>Tk_BindingTable bindingTable</strong> (in)</td>
<td>Token for binding table; must have been returned by some previous call to Tk_CreateBindingTable.</td>
</tr>
<tr>
<td><strong>ClientData object</strong> (in)</td>
<td>Identifies object with which binding is associated.</td>
</tr>
<tr>
<td>*<em>const char <em>eventString</em></em> (in)</td>
<td>String describing event sequence.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>char *script (in)</td>
<td>Tcl script to invoke when binding triggers.</td>
</tr>
<tr>
<td>int append (in)</td>
<td>Non-zero means append *script to existing script for binding, if any; zero means replace existing script with new one.</td>
</tr>
<tr>
<td>XEvent *eventPtr (in)</td>
<td>X event to match against bindings in <em>bindingTable</em>.</td>
</tr>
<tr>
<td>Tk_Window tkwin (in)</td>
<td>Identifier for any window on the display where the event occurred. Used to find display-related information such as key maps.</td>
</tr>
<tr>
<td>int numObjects (in)</td>
<td>Number of object identifiers pointed to by <em>objectPtr</em>.</td>
</tr>
<tr>
<td>ClientData *objectPtr (in)</td>
<td>Points to an array of object identifiers: bindings will be considered for each of these objects in order from first to last.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

These procedures provide a general-purpose mechanism for creating and invoking bindings. Bindings are organized in terms of *binding tables*. A binding table consists of a collection of bindings plus a history of recent events. Within a binding table, bindings are associated with *objects*. The meaning of an object is defined by clients of the binding
package. For example, Tk keeps uses one binding table to hold all of the bindings created by the \texttt{bind} command. For this table, objects are pointers to strings such as window names, class names, or other binding tags such as \texttt{all}. Tk also keeps a separate binding table for each canvas widget, which manages bindings created by the canvas's \texttt{bind} widget command; within this table, an object is either a pointer to the internal structure for a canvas item or a \texttt{Tk\_Uid} identifying a tag.

The procedure \texttt{Tk\_CreateBindingTable} creates a new binding table and associates \texttt{interp} with it (when bindings in the table are invoked, the scripts will be evaluated in \texttt{interp}). \texttt{Tk\_CreateBindingTable} returns a token for the table, which must be used in calls to other procedures such as \texttt{Tk\_CreateBinding} or \texttt{Tk\_BindEvent}.

\texttt{Tk\_DeleteBindingTable} frees all of the state associated with a binding table. Once it returns the caller should not use the \texttt{bindingTable} token again.

\texttt{Tk\_CreateBinding} adds a new binding to an existing table. The \texttt{object} argument identifies the object with which the binding is to be associated, and it may be any one-word value. Typically it is a pointer to a string or data structure. The \texttt{eventString} argument identifies the event or sequence of events for the binding; see the documentation for the \texttt{bind} command for a description of its format. \texttt{script} is the Tcl script to be evaluated when the binding triggers. \texttt{append} indicates what to do if there already exists a binding for \texttt{object} and \texttt{eventString}: if \texttt{append} is zero then \texttt{script} replaces the old script; if \texttt{append} is non-zero then the new script is appended to the old one. \texttt{Tk\_CreateBinding} returns an X event mask for all the events associated with the bindings. This information may be useful to invoke \texttt{XSelectInput} to select relevant events, or to disallow the use of certain events in bindings. If an error occurred while creating the binding (e.g., \texttt{eventString} refers to a non-existent event), then 0 is returned and an error message is left in \texttt{interp-}\texttt{\_result}.

\texttt{Tk\_DeleteBinding} removes from \texttt{bindingTable} the binding given by \texttt{object} and \texttt{eventString}, if such a binding exists. \texttt{Tk\_DeleteBinding} always returns \texttt{TCL\_OK}. In some cases it may reset \texttt{interp-}\texttt{\_result} to
the default empty value.

**Tk_GetBinding** returns a pointer to the script associated with `eventString` and `object` in `bindingTable`. If no such binding exists then NULL is returned and an error message is left in `interp->result`.

**Tk_GetAllBindings** returns in `interp->result` a list of all the event strings for which there are bindings in `bindingTable` associated with `object`. If there are no bindings for `object` then an empty string is returned in `interp->result`.

**Tk_DeleteAllBindings** deletes all of the bindings in `bindingTable` that are associated with `object`.

**Tk_BindEvent** is called to process an event. It makes a copy of the event in an internal history list associated with the binding table, then it checks for bindings that match the event. **Tk_BindEvent** processes each of the objects pointed to by `objectPtr` in turn. For each object, it finds all the bindings that match the current event history, selects the most specific binding using the priority mechanism described in the documentation for **bind**, and invokes the script for that binding. If there are no matching bindings for a particular object, then the object is skipped. **Tk_BindEvent** continues through all of the objects, handling exceptions such as errors, **break**, and **continue** as described in the documentation for **bind**.

**KEYWORDS**

**binding, event, object, script**
NAME
Tk_Init, Tk_SafeInit - add Tk to an interpreter and make a new Tk application.

SYNOPSIS
#include <tk.h>
int Tk_Init(interp)
int Tk_SafeInit(interp)

ARGUMENTS

DESCRIPTION

KEYWORDS

NAME
Tk_Init, Tk_SafeInit - add Tk to an interpreter and make a new Tk application.

SYNOPSIS
#include <tk.h>
int
Tk_Init(interp)
int
Tk_SafeInit(interp)

ARGUMENTS

Tcl_Interp *interp (in) Interpreter in which to load Tk. Tk should not already be loaded in this interpreter.

DESCRIPTION

Tk_Init is the package initialization procedure for Tk. It is normally invoked by the Tcl_AppInit procedure for an application or by the load command. Tk_Init adds all of Tk’s commands to interp and creates a new Tk application, including its main window. If the initialization is successful Tk_Init returns TCL_OK; if there is an error it returns TCL_ERROR. Tk_Init also leaves a result or error message in interp->result.

If there is a variable argv in interp, Tk_Init treats the contents of this variable as a list of options for the new Tk application. The options may have any of the forms documented for the wish application (in fact, wish uses Tk_Init to process its command-line arguments).

Tk_SafeInit is identical to Tk_Init except that it removes all Tk commands that are considered unsafe. Those commands and the reasons for their exclusion are:

bell
   Continuous ringing of the bell is a nuisance.

clipboard
   A malicious script could replace the contents of the clipboard with the string “rm -r *” and lead to surprises when the contents of the clipboard are pasted.
grab
   Grab can be used to block the user from using any other applications.

menu
   Menus can be used to cover the entire screen and to steal input from the user.

selection
   See clipboard.

send
   Send can be used to cause unsafe interpreters to execute commands.

tk
   The tk command recreates the send command, which is unsafe.

tkwait
   Tkwait can block the containing process forever

toplevel
   Toplevels can be used to cover the entire screen and to steal input from the user.

wm
   If toplevels are ever allowed, wm can be used to remove decorations, move windows around, etc.

KEYWORDS

safe, application, initialization, load, main window

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME
Tk_CanvasTkwin, Tk_CanvasGetCoord,
Tk_CanvasDrawableCoords, Tk_CanvasSetStippleOrigin,
Tk_CanvasWindowCoords, Tk_CanvasEventuallyRedraw,
Tk_CanvasTagsOption - utility procedures for canvas type managers

SYNOPSIS
#include <tk.h>
Tk_Window
Tk_CanvasTkwin(canvas)
int
Tk_CanvasGetCoord(interp, canvas, string, doublePtr)
Tk_CanvasDrawableCoords(canvas, x, y, drawableXPtr, drawableYPtr)
Tk_CanvasSetStippleOrigin(canvas, gc)
Tk_CanvasWindowCoords(canvas, x, y, screenXPtr, screenYPtr)
Tk_CanvasEventuallyRedraw(canvas, x1, y1, x2, y2)
Tk_OptionParseProc *Tk_CanvasTagsParseProc;
Tk_OptionPrintProc *Tk_CanvasTagsPrintProc;

ARGUMENTS
DESCRIPTION
KEYWORDS

NAME
Tk_CanvasTkwin, Tk_CanvasGetCoord, Tk_CanvasDrawableCoords,
Tk_CanvasSetStippleOrigin, Tk_CanvasWindowCoords,
Tk_CanvasEventuallyRedraw, Tk_CanvasTagsOption - utility procedures for canvas type managers
**SYNOPSIS**

```c
#include <tk.h>
Tk_Window
Tk_CanvasTkwin(canvas)
int
Tk_CanvasGetCoord(interp, canvas, string, doublePtr)
Tk_CanvasDrawableCoords(canvas, x, y, drawableXPtr, drawableYPtr)
Tk_CanvasSetStippleOrigin(canvas, gc)
Tk_CanvasWindowCoords(canvas, x, y, screenXPtr, screenYPtr)
Tk_CanvasEventuallyRedraw(canvas, x1, y1, x2, y2)
Tk_OptionParseProc *Tk_CanvasTagsParseProc;
Tk_OptionPrintProc *Tk_CanvasTagsPrintProc;
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Canvas canvas (in)</td>
<td>A token that identifies a canvas widget.</td>
</tr>
<tr>
<td>Tcl_Interp *interp (in/out)</td>
<td>Interpreter to use for error reporting.</td>
</tr>
<tr>
<td>const char *string (in)</td>
<td>Textual description of a canvas coordinate.</td>
</tr>
<tr>
<td>double *doublePtr (out)</td>
<td>Points to place to store a converted coordinate.</td>
</tr>
<tr>
<td>double x (in)</td>
<td>An x coordinate in the space of the canvas.</td>
</tr>
<tr>
<td>double y (in)</td>
<td>A y coordinate in the space of the canvas.</td>
</tr>
<tr>
<td>short *drawableXPtr (out)</td>
<td>Pointer to a location in which to store an x</td>
</tr>
</tbody>
</table>
A coordinate in the space of the drawable currently being used to redisplay the canvas.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>short *drawableYPtr (out)</td>
<td>Pointer to a location in which to store a y coordinate in the space of the drawable currently being used to redisplay the canvas.</td>
</tr>
<tr>
<td>GC gc (out)</td>
<td>Graphics context to modify.</td>
</tr>
<tr>
<td>short *screenXPtr (out)</td>
<td>Points to a location in which to store the screen coordinate in the canvas window that corresponds to x.</td>
</tr>
<tr>
<td>short *screenYPtr (out)</td>
<td>Points to a location in which to store the screen coordinate in the canvas window that corresponds to y.</td>
</tr>
<tr>
<td>int x1 (in)</td>
<td>Left edge of the region that needs redisplay. Only pixels at or to the right of this coordinate need to be redisplayed.</td>
</tr>
<tr>
<td>int y1 (in)</td>
<td>Top edge of the region that needs redisplay. Only pixels at or below this coordinate need to be redisplayed.</td>
</tr>
</tbody>
</table>
These procedures are called by canvas type managers to perform various utility functions.

**Tk_CanvasTkwin** returns the Tk_Window associated with a particular canvas.

**Tk_CanvasGetCoord** translates a string specification of a coordinate (such as 2p or 1.6c) into a double-precision canvas coordinate. If *string* is a valid coordinate description then **Tk_CanvasGetCoord** stores the corresponding canvas coordinate at *doublePtr* and returns **TCL_OK**. Otherwise it stores an error message in *interp->result* and returns **TCL_ERROR**.

**Tk_CanvasDrawableCoords** is called by type managers during redisplay to compute where to draw things. Given *x* and *y* coordinates in the space of the canvas, **Tk_CanvasDrawableCoords** computes the corresponding pixel in the drawable that is currently being used for redisplay; it returns those coordinates in *drawableXPtr* and *drawableYPtr*. This procedure should not be invoked except during redisplay.
Tk_CanvasSetStippleOrigin is also used during redisplay. It sets the stipple origin in gc so that stipples drawn with gc in the current offscreen pixmap will line up with stipples drawn with origin (0,0) in the canvas's actual window. Tk_CanvasSetStippleOrigin is needed in order to guarantee that stipple patterns line up properly when the canvas is redisplayed in small pieces. Redisplays are carried out in double-buffered fashion where a piece of the canvas is redrawn in an offscreen pixmap and then copied back onto the screen. In this approach the stipple origins in graphics contexts need to be adjusted during each redisplay to compensate for the position of the off-screen pixmap relative to the window. If an item is being drawn with stipples, its type manager typically calls Tk_CanvasSetStippleOrigin just before using gc to draw something; after it is finished drawing, the type manager calls XSetTSOrigin to restore the origin in gc back to (0,0) (the restore is needed because graphics contexts are shared, so they cannot be modified permanently).

Tk_CanvasWindowCoords is similar to Tk_CanvasDrawableCoords except that it returns coordinates in the canvas's window on the screen, instead of coordinates in an off-screen pixmap.

Tk_CanvasEventuallyRedraw may be invoked by a type manager to inform Tk that a portion of a canvas needs to be redrawn. The x1, y1, x2, and y2 arguments specify the region that needs to be redrawn, in canvas coordinates. Type managers rarely need to invoke Tk_CanvasEventuallyRedraw, since Tk can normally figure out when an item has changed and make the redisplay request on its behalf (this happens, for example whenever Tk calls a configureProc or scaleProc). The only time that a type manager needs to call Tk_CanvasEventuallyRedraw is if an item has changed on its own without being invoked through one of the procedures in its Tk_ItemType; this could happen, for example, in an image item if the image is modified using image commands.

Tk_CanvasTagsParseProc and Tk_CanvasTagsPrintProc are procedures that handle the -tags option for canvas items. The code of a canvas type manager will not call these procedures directly, but will use their addresses to create a Tk_CustomOption structure for the -tags
option. The code typically looks like this:

```c
static Tk_CustomOption tagsOption = {Tk_CanvasTagsParseProc, Tk_CanvasTagsPrintProc, (ClientData) NULL
};

static Tk_ConfigSpec configSpecs[] = {
    ...
    {TK_CONFIG_CUSTOM, "-tags", (char *) NULL, (char *) NULL, (char *) NULL, 0, TK_CONFIG_NULL_OK, &tagsOption},
    ...
};
```

**KEYWORDS**

canvas, focus, item type, redisplay, selection, type manager

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_HWNDToWindow - Find Tk's window information for a Windows window

SYNOPSIS

#include <tkPlatDecIs.h>
Tk_Window
Tk_HWNDToWindow(hwnd)

ARGUMENTS

HWND hwnd (in)  Windows handle for the window.

DESCRIPTION

Given a Windows HWND window identifier, this procedure returns the corresponding Tk_Window handle. If there is no Tk_Window corresponding to hwnd then NULL is returned.

KEYWORDS

Windows window id

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

Tk_IdToWindow - Find Tk's window information for an X window

SYNOPSIS

#include <tk.h>
Tk_Window
Tk_IdToWindow(display, window)

ARGUMENTS

Display *display (in) X display containing the window.
Window window (in) X id for window.

DESCRIPTION

Given an X window identifier and the X display it corresponds to, this
procedure returns the corresponding Tk_Window handle. If there is no
Tk_Window corresponding to window then NULL is returned.

KEYWORDS

X window id

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME

Tk_ImageChanged - notify widgets that image needs to be redrawn

SYNOPSIS

```
#include <tk.h>
Tk_ImageChanged(imageMaster, x, y, width, height, imageWidth, imageHeight)
```

ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_ImageMaster imageMaster (in)</td>
<td>Token for image, which was passed to image's createProc when the image was created.</td>
</tr>
<tr>
<td>int x (in)</td>
<td>X-coordinate of upper-left corner of region that needs redisplay (measured from upper-left corner of image).</td>
</tr>
<tr>
<td>int y (in)</td>
<td>Y-coordinate of upper-left corner of region that needs redisplay (measured from upper-left corner of image).</td>
</tr>
<tr>
<td>int width (in)</td>
<td>Width of region that needs to be redrawn, in pixels.</td>
</tr>
</tbody>
</table>
### DESCRIPTION

An image manager calls **Tk_ImageChanged** for an image whenever anything happens that requires the image to be redrawn. As a result of calling **Tk_ImageChanged**, any widgets using the image are notified so that they can redisplay themselves appropriately. The *imageMaster* argument identifies the image, and *x*, *y*, *width*, and *height* specify a rectangular region within the image that needs to be redrawn. *imageWidth* and *imageHeight* specify the image's (new) size.

An image manager should call **Tk_ImageChanged** during its *createProc* to specify the image's initial size and to force redisplay if there are existing instances for the image. If any of the pixel values in the image should change later on, **Tk_ImageChanged** should be called again with *x*, *y*, *width*, and *height* values that cover all the pixels that changed. If the size of the image should change, then **Tk_ImageChanged** must be called to indicate the new size, even if no pixels need to be redisplayed.

### SEE ALSO

**Tk_CreateImageType**

### KEYWORDS

images, redisplay, image size changes
**NAME**

Tk_SetAppName - Set the name of an application for 'send' commands

**SYNOPSIS**

```c
#include <tk.h>
const char *
Tk_SetAppName(tkwin, name)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Window tkwin</td>
<td>Token for window in application. Used only to select a particular application.</td>
</tr>
<tr>
<td>const char *name</td>
<td>Name under which to register the application.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

**Tk_SetAppName** associates a name with a given application and records that association on the display containing with the application's main window. After this procedure has been invoked, other applications on the display will be able to use the **send** command to invoke operations in the application. If *name* is already in use by some other application on the display, then a new name will be generated by appending " #2" to *name*; if this name is also in use, the number will be incremented until an unused name is found. The return value from the procedure is a pointer to the name actually used.
If the application already has a name when `Tk_SetAppName` is called, then the new name replaces the old name.

`Tk_SetAppName` also adds a `send` command to the application's interpreter, which can be used to send commands from this application to others on any of the displays where the application has windows.

The application's name registration persists until the interpreter is deleted or the `send` command is deleted from `interp`, at which point the name is automatically unregistered and the application becomes inaccessible via `send`. The application can be made accessible again by calling `Tk_SetAppName`.

`Tk_SetAppName` is called automatically by `Tk_Init`, so applications do not normally need to call it explicitly.

The command `tk appname` provides Tcl-level access to the functionality of `Tk_SetAppName`.

**KEYWORDS**

application, name, register, send command
NAME
Tk_CanvasPsY, Tk_CanvasPsBitmap, Tk_CanvasPsColor, Tk_CanvasPsFont, Tk_CanvasPsPath, Tk_CanvasPsStipple - utility procedures for generating Postscript for canvases

SYNOPSIS
#include <tk.h>
double
Tk_CanvasPsY(canvas, canvasY)
int
Tk_CanvasPsBitmap(interp, canvas, bitmap, x, y, width, height)
int
Tk_CanvasPsColor(interp, canvas, colorPtr)
int
Tk_CanvasPsFont(interp, canvas, tkFont)
Tk_CanvasPsPath(interp, canvas, coordPtr, numPoints)
int
Tk_CanvasPsStipple(interp, canvas, bitmap)

ARGUMENTS
DESCRIPTION
KEYWORDS
Tk\_CanvasPsY*(canvas, canvasY)*
int

Tk\_CanvasPsBitmap*(interp, canvas, bitmap, x, y, width, height)*
int

Tk\_CanvasPsColor*(interp, canvas, colorPtr)*
int

Tk\_CanvasPsFont*(interp, canvas, tkFont)*

Tk\_CanvasPsPath*(interp, canvas, coordPtr, numPoints)*
int

Tk\_CanvasPsStipple*(interp, canvas, bitmap)*

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Canvas <em>canvas</em> (in)</td>
<td>A token that identifies a canvas widget for which Postscript is being generated.</td>
</tr>
<tr>
<td>double <em>canvasY</em> (in)</td>
<td>Y-coordinate in the space of the canvas.</td>
</tr>
<tr>
<td><em>Tcl_Interp</em> <em>interp</em> (in/out)</td>
<td>A Tcl interpreter; Postscript is appended to its result, or the result may be replaced with an error message.</td>
</tr>
<tr>
<td>Pixmap <em>bitmap</em> (in)</td>
<td>Bitmap to use for generating Postscript.</td>
</tr>
<tr>
<td>int <em>x</em> (in)</td>
<td>X-coordinate within <em>bitmap</em> of left edge of region to output.</td>
</tr>
<tr>
<td>int <em>y</em> (in)</td>
<td>Y-coordinate within <em>bitmap</em> of top edge of region to output.</td>
</tr>
</tbody>
</table>
int width (in) Width of region of bitmap to output, in pixels.

int height (in) Height of region of bitmap to output, in pixels.

XColor *colorPtr (in) Information about color value to set in Postscript.

Tk_Font tkFont (in) Font for which Postscript is to be generated.

double *coordPtr (in) Pointer to an array of coordinates for one or more points specified in canvas coordinates. The order of values in coordPtr is x1, y1, x2, y2, x3, y3, and so on.

int numPoints (in) Number of points at coordPtr.

DESCRIPTION

These procedures are called by canvas type managers to carry out common functions related to generating Postscript. Most of the procedures take a canvas argument, which refers to a canvas widget for which Postscript is being generated.

Tk_CanvasPsY takes as argument a y-coordinate in the space of a canvas and returns the value that should be used for that point in the Postscript currently being generated for canvas. Y coordinates require transformation because Postscript uses an origin at the lower-left corner whereas X uses an origin at the upper-left corner. Canvas x coordinates can be used directly in Postscript without transformation.
**Tk_CanvasPsBitmap** generates Postscript to describe a region of a bitmap. The Postscript is generated in proper image data format for Postscript, i.e., as data between angle brackets, one bit per pixel. The Postscript is appended to `interp->result` and TCL_OK is returned unless an error occurs, in which case TCL_ERROR is returned and `interp->result` is overwritten with an error message.

**Tk_CanvasPsColor** generates Postscript to set the current color to correspond to its `colorPtr` argument, taking into account any color map specified in the `postscript` command. It appends the Postscript to `interp->result` and returns TCL_OK unless an error occurs, in which case TCL_ERROR is returned and `interp->result` is overwritten with an error message.

**Tk_CanvasPsFont** generates Postscript that sets the current font to match `tkFont` as closely as possible. **Tk_CanvasPsFont** takes into account any font map specified in the `postscript` command, and it does the best it can at mapping X fonts to Postscript fonts. It appends the Postscript to `interp->result` and returns TCL_OK unless an error occurs, in which case TCL_ERROR is returned and `interp->result` is overwritten with an error message.

**Tk_CanvasPsPath** generates Postscript to set the current path to the set of points given by `coordPtr` and `numPoints`. It appends the resulting Postscript to `interp->result`.

**Tk_CanvasPsStipple** generates Postscript that will fill the current path in stippled fashion. It uses `bitmap` as the stipple pattern and the current Postscript color; ones in the stipple bitmap are drawn in the current color, and zeroes are not drawn at all. The Postscript is appended to `interp->result` and TCL_OK is returned, unless an error occurs, in which case TCL_ERROR is returned and `interp->result` is overwritten with an error message.

**KEYWORDS**

bitmap, canvas, color, font, path, Postscript, stipple
NAME

Tk_InitConsoleChannels - Install the console channels as standard channels

SYNOPSIS

#include <tk.h>
Tk_InitConsoleChannels(interp)

ARGUMENTS

Tcl_Interp *interp (in) Interpreter in which the console channels are created.

DESCRIPTION

Tk_InitConsoleChannels is invoked to create a set of console channels and install them as the standard channels. All I/O on these channels will be discarded until Tk_CreateConsoleWindow is called to attach the console to a text widget.

This function is for use by shell applications based on Tk, like wish, on platforms which have no standard channels in graphical mode, like Win32.

The interp argument is the interpreter in which to create and install the console channels.

NOTE: If this function is used it has to be called before the first call to
Tcl_RegisterChannel, directly, or indirectly through other channel functions. Because otherwise the standard channels will be already initialized to the system defaults, which will be nonsensical for the case Tk_InitConsoleChannels is for.

SEE ALSO

console

KEYWORDS

standard channels, console

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2007 ActiveState Software Inc.
NAME

Tk_SetCaretPos - set the display caret location

SYNOPSIS

#include <tk.h>
int
Tk_SetCaretPos(tkwin, x, y, height)

ARGUMENTS

Tk_Window tkwin (in)         Token for window.
int x (in)                   Window-relative x coordinate.
int y (in)                   Window-relative y coordinate.
int h (in)                   Height of the caret in the window.

DESCRIPTION

Tk_SetCaretPos sets the caret location for the display of the specified Tk_Window tkwin. The caret is the per-display cursor location used for indicating global focus (e.g. to comply with Microsoft Accessibility guidelines), as well as for location of the over-the-spot XIM (X Input Methods) or Windows IME windows.
<table>
<thead>
<tr>
<th>KEYWORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>caret, cursor</td>
</tr>
</tbody>
</table>

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2002 ActiveState Corporation.
NAME
Tk_CreateImageType, Tk_GetImageMasterData, Tk_InitImageArgs - define new kind of image

SYNOPSIS
#include <tk.h>
Tk_CreateImageType(typePtr)
ClientData
Tk_GetImageMasterData(interp, name, typePtrPtr)
Tk_InitImageArgs(interp, argc, argvPtr)

ARGUMENTS
DESCRIPTION
NAME
CREATEPROC
GETPROC
DISPLAYPROC
FREEPROC
DELETEPROC
TK_GETIMAGEMASTERDATA
LEGACY INTERFACE SUPPORT
SEE ALSO
KEYWORDS

NAME
Tk_CreateImageType, Tk_GetImageMasterData, Tk_InitImageArgs - define new kind of image

SYNOPSIS
#include <tk.h>
Tk_CreateImageType(typePtr)
ClientData
**Tk_GetImageMasterData**(*interp, name, typePtrPtr*)

**Tk_InitImageArgs**(*interp, argc, argvPtr*)

## ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_ImageType *typePtr (in)</td>
<td>Structure that defines the new type of image. Must be static: a pointer to this structure is retained by the image code.</td>
</tr>
<tr>
<td>Tcl_Interp *interp (in)</td>
<td>Interpreter in which image was created.</td>
</tr>
<tr>
<td>const char *name (in)</td>
<td>Name of existing image.</td>
</tr>
<tr>
<td>Tk_ImageType **typePtrPtr (out)</td>
<td>Points to word in which to store a pointer to type information for the given image, if it exists.</td>
</tr>
<tr>
<td>int argc (in)</td>
<td>Number of arguments</td>
</tr>
<tr>
<td>char ***argvPtr (in/out)</td>
<td>Pointer to argument list</td>
</tr>
</tbody>
</table>

## DESCRIPTION

**Tk_CreateImageType** is invoked to define a new kind of image. An image type corresponds to a particular value of the type argument for the image create command. There may exist any number of different image types, and new types may be defined dynamically by calling **Tk_CreateImageType**. For example, there might be one type for 2-color bitmaps, another for multi-color images, another for dithered images, another for video, and so on.

The code that implements a new image type is called an *image*
manager. It consists of a collection of procedures plus three different kinds of data structures. The first data structure is a Tk_ImageType structure, which contains the name of the image type and pointers to five procedures provided by the image manager to deal with images of this type:

```c
typedef struct Tk_ImageType {
    char *name;
    Tk_ImageCreateProc *createProc;
    Tk_ImageGetProc *getProc;
    Tk_ImageDisplayProc *displayProc;
    Tk_ImageFreeProc *freeProc;
    Tk_ImageDeleteProc *deleteProc;
} Tk_ImageType;
```

The fields of this structure will be described in later subsections of this entry.

The second major data structure manipulated by an image manager is called an image master; it contains overall information about a particular image, such as the values of the configuration options specified in an image create command. There will usually be one of these structures for each invocation of the image create command.

The third data structure related to images is an image instance. There will usually be one of these structures for each usage of an image in a particular widget. It is possible for a single image to appear simultaneously in multiple widgets, or even multiple times in the same widget. Furthermore, different instances may be on different screens or displays. The image instance data structure describes things that may vary from instance to instance, such as colors and graphics contexts for redisplay. There is usually one instance structure for each -image option specified for a widget or canvas item.

The following subsections describe the fields of a Tk_ImageType in more detail.
**NAME**

`typePtr->name` provides a name for the image type. Once `Tk_CreateImageType` returns, this name may be used in `image create` commands to create images of the new type. If there already existed an image type by this name then the new image type replaces the old one.

**CREATEPROC**

`typePtr->createProc` provides the address of a procedure for Tk to call whenever `image create` is invoked to create an image of the new type. `typePtr->createProc` must match the following prototype:

```c
typedef int Tk_ImageCreateProc(
    Tcl_Interp *interp,
    char *name, 
    int objc,
    Tcl_Obj *const objv[],
    Tk_ImageType *typePtr, 
    Tk_ImageMaster master, 
    ClientData *masterDataPtr);
```

The `interp` argument is the interpreter in which the `image` command was invoked, and `name` is the name for the new image, which was either specified explicitly in the `image` command or generated automatically by the `image` command. The `objc` and `objv` arguments describe all the configuration options for the new image (everything after the name argument to `image`). The `master` argument is a token that refers to Tk’s information about this image; the image manager must return this token to Tk when invoking the `Tk_ImageChanged` procedure. Typically `createProc` will parse `objc` and `objv` and create an image master data structure for the new image. `createProc` may store an arbitrary one-word value at `*masterDataPtr`, which will be passed back to the image manager when other callbacks are invoked. Typically the value is a pointer to the master data structure for the image.
If `createProc` encounters an error, it should leave an error message in the interpreter result and return `TCL_ERROR`; otherwise it should return `TCL_OK`.

`createProc` should call `Tk_ImageChanged` in order to set the size of the image and request an initial redisplay.

**GETPROC**

typePtr->`getProc` is invoked by Tk whenever a widget calls `Tk_GetImage` to use a particular image. This procedure must match the following prototype:

```c
typedef ClientData Tk_ImageGetProc(
    Tk_Window tkwin,
    ClientData masterData);
```

The `tkwin` argument identifies the window in which the image will be used and `masterData` is the value returned by `createProc` when the image master was created. `getProc` will usually create a data structure for the new instance, including such things as the resources needed to display the image in the given window. `getProc` returns a one-word token for the instance, which is typically the address of the instance data structure. Tk will pass this value back to the image manager when invoking its `displayProc` and `freeProc` procedures.

**DISPLAYPROC**

typePtr->`displayProc` is invoked by Tk whenever an image needs to be displayed (i.e., whenever a widget calls `Tk_RedrawImage`). `displayProc` must match the following prototype:

```c
typedef void Tk_ImageDisplayProc(
    ClientData instanceData,  
    Display *display,  
);  
```
Drawable drawable,
int imageX,
int imageY,
int width,
int height,
int drawableX,
int drawableY);

The instanceData will be the same as the value returned by getProc when the instance was created. display and drawable indicate where to display the image; drawable may be a pixmap rather than the window specified to getProc (this is usually the case, since most widgets double-buffer their redisplay to get smoother visual effects). imageX, imageY, width, and height identify the region of the image that must be redisplayed. This region will always be within the size of the image as specified in the most recent call to Tk_ImageChanged. drawableX and drawableY indicate where in drawable the image should be displayed; displayProc should display the given region of the image so that point (imageX, imageY) in the image appears at (drawableX, drawableY) in drawable.

FREEPROC

typePtr->freeProc contains the address of a procedure that Tk will invoke when an image instance is released (i.e., when Tk_FreeImage is invoked). This can happen, for example, when a widget is deleted or a image item in a canvas is deleted, or when the image displayed in a widget or canvas item is changed. freeProc must match the following prototype:

typedef void Tk_ImageFreeProc(
    ClientData instanceData,
    Display *display);

The instanceData will be the same as the value returned by getProc
when the instance was created, and *display is the display containing the window for the instance. freeProc should release any resources associated with the image instance, since the instance will never be used again.

**DELETEPROC**

typePtr->deleteProc is a procedure that Tk invokes when an image is being deleted (i.e. when the *image delete* command is invoked). Before invoking deleteProc Tk will invoke freeProc for each of the image's instances. deleteProc must match the following prototype:

```c
typedef void Tk_ImageDeleteProc(
    ClientData masterData);
```

The masterData argument will be the same as the value stored in *masterDataPtr by createProc when the image was created. deleteProc should release any resources associated with the image.

**TK_GETIMAGEMASTERDATA**

The procedure Tk_GetImageMasterData may be invoked to retrieve information about an image. For example, an image manager can use this procedure to locate its image master data for an image. If there exists an image named name in the interpreter given by interp, then *typePtrPtr is filled in with type information for the image (the typePtr value passed to Tk_CreateImageType when the image type was registered) and the return value is the ClientData value returned by the createProc when the image was created (this is typically a pointer to the image master data structure). If no such image exists then NULL is returned and NULL is stored at *typePtrPtr.

**LEGACY INTERFACE SUPPORT**

In Tk 8.2 and earlier, the definition of Tk_ImageCreateProc was incompatibly different, with the following prototype:
typedef int Tk_ImageCreateProc(
   Tcl_Interp *interp,
   char *name,
   int argc,
   char **argv,
   Tk_ImageType *typePtr,
   Tk_ImageMaster master,
   ClientData *masterDataPtr);

Legacy programs and libraries dating from those days may still contain code that defines extended Tk image types using the old interface. The Tk header file will still support this legacy interface if the code is compiled with the macro `USE_OLD_IMAGE` defined.

When the `USE_OLD_IMAGE` legacy support is enabled, you may see the routine `Tk_InitImageArgs` in use. This was a migration tool used to create stub-enabled extensions that could be loaded into interps containing all versions of Tk 8.1 and later. Tk 8.5 no longer provides this routine, but uses a macro to convert any attempted calls of this routine into an empty comment. Any stub-enabled extension providing an extended image type via the legacy interface that is compiled against Tk 8.5 headers and linked against the Tk 8.5 stub library will produce a file that can be loaded only into interps with Tk 8.5 or later; that is, the normal stub-compatibility rules. If a developer needs to generate from such code a file that is loadable into interps with Tk 8.4 or earlier, they must use Tk 8.4 headers and stub libraries to do so.

Any new code written today should not make use of the legacy interfaces. Expect their support to go away in Tk 9.

**SEE ALSO**

`Tk_ImageChanged`, `Tk_GetImage`, `Tk_FreeImage`, `Tk_RedrawImage`, `Tk_SizeOfImage`

**KEYWORDS**
image manager, image type, instance, master

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1997 Sun Microsystems, Inc.
NAME

Tk_SetClass, Tk_Class - set or retrieve a window's class

SYNOPSIS

#include <tk.h>
Tk_SetClass(tkwin, class)
Tk_Uid
Tk_Class(tkwin)

ARGUMENTS

Tk_Window tkwin (in) Token for window.

char *class (in) New class name for window.

DESCRIPTION

Tk_SetClass is called to associate a class with a particular window. The class string identifies the type of the window; all windows with the same general class of behavior (button, menu, etc.) should have the same class. By convention all class names start with a capital letter, and there exists a Tcl command with the same name as each class (except all in lower-case) which can be used to create and manipulate windows of that class. A window's class string is initialized to NULL when the window is created.

For main windows, Tk automatically propagates the name and class to the WM_CLASS property used by window managers. This happens
either when a main window is actually created (e.g. in \texttt{Tk\_MakeWindowExist}), or when \texttt{Tk\_SetClass} is called, whichever occurs later. If a main window has not been assigned a class then Tk will not set the WM\_CLASS property for the window.

\texttt{Tk\_Class} is a macro that returns the current value of \texttt{tkwin}'s class. The value is returned as a \texttt{Tk\_Uid}, which may be used just like a string pointer but also has the properties of a unique identifier (see the manual entry for \texttt{Tk\_GetUid} for details). If \texttt{tkwin} has not yet been given a class, then \texttt{Tk\_Class} will return NULL.

\textbf{KEYWORDS}

\texttt{class, unique identifier, window, window manager}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_SetClassProcs - register widget specific procedures

SYNOPSIS

#include <tk.h>
Tk_SetClassProcs(tkwin, procs, instanceData)

ARGUMENTS

Tk_Window tkwin (in)          Token for window to modify.
Tk_ClassProcs *procs (in)     Pointer to data structure containing widget specific procedures. The data structure pointed to by procs must be static: Tk keeps a reference to it as long as the window exists.
ClientData instanceData (in)  Arbitrary one-word value to pass to widget callbacks.

DESCRIPTION

Tk_SetClassProcs is called to register a set of procedures that are used as callbacks in different places.
The structure pointed to by `procs` contains the following:

```c
typedef struct Tk_ClassProcs {
    unsigned int size;
    Tk_ClassWorldChangedProc *worldChangedProc;
    Tk_ClassCreateProc *createProc;
    Tk_ClassModalProc *modalProc;
} Tk_ClassProcs;
```

The `size` field is used to simplify future expansion of the structure. It should always be set to (literally) `sizeof(Tk_ClassProcs)`.

`worldChangedProc` is invoked when the system has altered in some way that requires some reaction from the widget. For example, when a font alias (see the `font` manual entry) is reconfigured, widgets configured to use that font alias must update their display accordingly. `worldChangedProc` should have arguments and results that match the type `Tk_ClassWorldChangedProc`:

```c
typedef void Tk_ClassWorldChangedProc(
    ClientData instanceData);
```

The `instanceData` parameter passed to the `worldChangedProc` will be identical to the `instanceData` parameter passed to `Tk_SetClassProcs`.

`createProc` is used to create platform-dependant windows. It is invoked by `Tk_MakeWindowExist`. `createProc` should have arguments and results that match the type `Tk_ClassCreateProc`:

```c
typedef Window Tk_ClassCreateProc(
    Tk_Window tkwin,
    Window parent,
    ClientData instanceData);
```
The `tkwin` and `instanceData` parameters will be identical to the `tkwin` and `instanceData` parameters passed to `Tk_SetClassProcs`. The `parent` parameter will be the parent of the window to be created. The `createProc` should return the created window.

`modalProc` is invoked after all bindings on a widget have been triggered in order to handle a modal loop. `modalProc` should have arguments and results that match the type `Tk_ClassModalProc`:

```c
typedef void Tk_ClassModalProc(
    Tk_Window tkwin,
    XEvent *eventPtr);
```

The `tkwin` parameter to `modalProc` will be identical to the `tkwin` parameter passed to `Tk_SetClassProcs`. The `eventPtr` parameter will be a pointer to an XEvent structure describing the event being processed.

**KEYWORDS**

callback, class
NAME
Tk_InitStubs - initialize the Tk stubs mechanism

SYNOPSIS
#include <tk.h>
const char *
            Tk_InitStubs(interp, version, exact)

ARGUMENTS
Tcl_Interpreter *interp (in)
A version string consisting of one or more decimal
The Tcl stubs mechanism defines a way to dynamically bind extensions to a particular Tcl implementation at run time. The stubs mechanism requires no changes to applications incorporating Tcl/Tk interpreters. Only developers creating C-based Tcl/Tk extensions need to take steps to use the stubs mechanism with their extensions. See the `Tcl_InitStubs` page for more information.

Enabling the stubs mechanism for a Tcl/Tk extension requires the following steps:

1) Call `Tcl_InitStubs` in the extension before calling any other Tcl functions.

2) Call `Tk_InitStubs` if the extension before calling any other Tk functions.

Define the `USE_TCL_STUBS` symbol. Typically, you would include the `-DUSE_TCL_STUBS` flag when compiling the extension.
3) Link the extension with the Tcl and Tk stubs libraries instead of the standard Tcl and Tk libraries. On Unix platforms, the library names are libtclstub8.4.a and libtkstub8.4.a; on Windows platforms, the library names are tclstub84.lib and tkstub84.lib (adjust names with appropriate version number).

**DESCRIPTION**

**Tk_InitStubs** attempts to initialize the Tk stub table pointers and ensure that the correct version of Tk is loaded. In addition to an interpreter handle, it accepts as arguments a version number and a Boolean flag indicating whether the extension requires an exact version match or not. If `exact` is 0, then the extension is indicating that newer versions of Tk are acceptable as long as they have the same major version number as `version`; non-zero means that only the specified `version` is acceptable. **Tcl_InitStubs** returns a string containing the actual version of Tk satisfying the request, or NULL if the Tk version is not acceptable, does not support the stubs mechanism, or any other error condition occurred.

**SEE ALSO**

[Tcl_InitStubs](#)

**KEYWORDS**

stubs
**NAME**

Tk_SetGrid, Tk_UnsetGrid - control the grid for interactive resizing

**SYNOPSIS**

```c
#include <tk.h>
Tk_SetGrid(tkwin, reqWidth, reqHeight, widthInc, heightInc)
Tk_UnsetGrid(tkwin)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Window tkwin (in)</td>
<td>Token for window.</td>
</tr>
<tr>
<td>int reqWidth (in)</td>
<td>Width in grid units that corresponds to the pixel dimension <code>tkwin</code> has requested via <code>Tk_GeometryRequest</code>.</td>
</tr>
<tr>
<td>int reqHeight (in)</td>
<td>Height in grid units that corresponds to the pixel dimension <code>tkwin</code> has requested via <code>Tk_GeometryRequest</code>.</td>
</tr>
<tr>
<td>int widthInc (in)</td>
<td>Width of one grid unit, in pixels.</td>
</tr>
<tr>
<td>int heightInc (in)</td>
<td>Height of one grid unit, in pixels.</td>
</tr>
</tbody>
</table>
Tk_SetGrid turns on gridded geometry management for tkwin's toplevel window and specifies the geometry of the grid. Tk_SetGrid is typically invoked by a widget when its setGrid option is true. It restricts interactive resizing of tkwin's toplevel window so that the space allocated to the toplevel is equal to its requested size plus or minus even multiples of widthInc and heightInc. Furthermore, the reqWidth and reqHeight values are passed to the window manager so that it can report the window's size in grid units during interactive resizes. If tkwin's configuration changes (e.g., the size of a grid unit changes) then the widget should invoke Tk_SetGrid again with the new information.

Tk_UnsetGrid cancels gridded geometry management for tkwin's toplevel window.

For each toplevel window there can be at most one internal window with gridding enabled. If Tk_SetGrid or Tk_UnsetGrid is invoked when some other window is already controlling gridding for tkwin's toplevel, the calls for the new window have no effect.

See the wmm manual entry for additional information on gridded geometry management.

**KEYWORDS**

grid, window, window manager

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tk_GeometryRequest, Tk_SetMinimumRequestSize, Tk_SetInternalBorder, Tk_SetInternalBorderEx - specify desired geometry or internal border for a window

SYNOPSIS
#include <tk.h>
Tk_GeometryRequest(tkwin, reqWidth, reqHeight)
Tk_SetMinimumRequestSize(tkwin, minWidth, minHeight)
Tk_SetInternalBorder(tkwin, width)
Tk_SetInternalBorderEx(tkwin, left, right, top, bottom)

ARGUMENTS
Tk_Window tkwin (in)  Window for which geometry is being
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int reqWidth (in)</td>
<td>Desired width for <em>tkwin</em>, in pixel units.</td>
</tr>
<tr>
<td>int reqHeight (in)</td>
<td>Desired height for <em>tkwin</em>, in pixel units.</td>
</tr>
<tr>
<td>int minWidth (in)</td>
<td>Desired minimum requested width for <em>tkwin</em>, in pixel units.</td>
</tr>
<tr>
<td>int minHeight (in)</td>
<td>Desired minimum requested height for <em>tkwin</em>, in pixel units.</td>
</tr>
<tr>
<td>int width (in)</td>
<td>Space to leave for internal border for <em>tkwin</em>, in pixel units.</td>
</tr>
<tr>
<td>int left (in)</td>
<td>Space to leave for left side of internal border for <em>tkwin</em>, in pixel units.</td>
</tr>
<tr>
<td>int right (in)</td>
<td>Space to leave for right side of internal border for <em>tkwin</em>, in pixel units.</td>
</tr>
<tr>
<td>int top (in)</td>
<td>Space to leave for top side of internal border for <em>tkwin</em>, in pixel units.</td>
</tr>
<tr>
<td>int bottom (in)</td>
<td>Space to leave for bottom side of internal border for <em>tkwin</em>, in pixel units.</td>
</tr>
</tbody>
</table>
**DESCRIPTION**

**Tk_GeometryRequest** is called by widget code to indicate its preference for the dimensions of a particular window. The arguments to **Tk_GeometryRequest** are made available to the geometry manager for the window, which then decides on the actual geometry for the window. Although geometry managers generally try to satisfy requests made to **Tk_GeometryRequest**, there is no guarantee that this will always be possible. Widget code should not assume that a geometry request will be satisfied until it receives a **ConfigureNotify** event indicating that the geometry change has occurred. Widget code should never call procedures like **Tk_ResizeWindow** directly. Instead, it should invoke **Tk_GeometryRequest** and leave the final geometry decisions to the geometry manager.

If **tkwin** is a top-level window, then the geometry information will be passed to the window manager using the standard ICCCM protocol.

**Tk_SetInternalBorder** is called by widget code to indicate that the widget has an internal border. This means that the widget draws a decorative border inside the window instead of using the standard X borders, which are external to the window's area. For example, internal borders are used to draw 3-D effects. **Width** specifies the width of the border in pixels. Geometry managers will use this information to avoid placing any children of **tkwin** overlapping the outermost **width** pixels of **tkwin**'s area.

**Tk_SetInternalBorderEx** works like **Tk_SetInternalBorder** but lets you specify different widths for different sides of the window.

**Tk_SetMinimumRequestSize** is called by widget code to indicate that a geometry manager should request at least this size for the widget. This allows a widget to have some control over its size when a propagating geometry manager is used inside it.

The information specified in calls to **Tk_GeometryRequest**, **Tk_SetMinimumRequestSize**, **Tk_SetInternalBorder** and **Tk_SetInternalBorderEx** can be retrieved using the macros
Tk ReqWidth, Tk ReqHeight, Tk MinReqWidth, Tk MinReqHeight, Tk MinReqWidth, Tk InternalBorderLeft, Tk InternalBorderRight, Tk InternalBorderTop and Tk InternalBorderBottom. See the Tk WindowId manual entry for details.

**KEYWORDS**

geometry, request
NAME

Tk_GetGC, Tk_FreeGC - maintain database of read-only graphics contexts

SYNOPSIS

```c
#include <tk.h>
GC
Tk_GetGC(tkwin, valueMask, valuePtr)
Tk_FreeGC(display, gc)
```

ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Window <code>tkwin</code> (in)</td>
<td>Token for window in which the graphics context will be used.</td>
</tr>
<tr>
<td><code>unsigned long valueMask</code> (in)</td>
<td>Mask of bits (such as <code>GCForeground</code> or <code>GCStipple</code>) indicating which fields of <code>valuePtr</code> are valid.</td>
</tr>
<tr>
<td><code>XGCValues *valuePtr</code> (in)</td>
<td>Pointer to structure describing the desired values for the graphics context.</td>
</tr>
<tr>
<td><code>Display *display</code> (in)</td>
<td>Display for which <code>gc</code> was allocated.</td>
</tr>
</tbody>
</table>
GC gc (in) X identifier for graphics context that is no longer needed. Must have been allocated by Tk_GetGC.

**DESCRIPTION**

Tk_GetGC and Tk_FreeGC manage a collection of graphics contexts being used by an application. The procedures allow graphics contexts to be shared, thereby avoiding the server overhead that would be incurred if a separate GC were created for each use. Tk_GetGC takes arguments describing the desired graphics context and returns an X identifier for a GC that fits the description. The graphics context that is returned will have default values in all of the fields not specified explicitly by valueMask and valuePtr.

Tk_GetGC maintains a database of all the graphics contexts it has created. Whenever possible, a call to Tk_GetGC will return an existing graphics context rather than creating a new one. This approach can substantially reduce server overhead, so Tk_GetGC should generally be used in preference to the Xlib procedure XCreateGC, which creates a new graphics context on each call.

Since the return values of Tk_GetGC are shared, callers should never modify the graphics contexts returned by Tk_GetGC. If a graphics context must be modified dynamically, then it should be created by calling XCreateGC instead of Tk_GetGC.

When a graphics context is no longer needed, Tk_FreeGC should be called to release it. There should be exactly one call to Tk_FreeGC for each call to Tk_GetGC. When a graphics context is no longer in use anywhere (i.e. it has been freed as many times as it has been gotten) Tk_FreeGC will release it to the X server and delete it from the database.

**KEYWORDS**
## graphics context

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_CanvasTextInfo - additional information for managing text items in canvases

SYNOPSIS

#include <tk.h>
Tk_CanvasTextInfo *
Tk_CanvasGetTextInfo(canvas)

ARGUMENTS

Tk_Canvas canvas (in) A token that identifies a particular canvas widget.

DESCRIPTION

Textual canvas items are somewhat more complicated to manage than other items, due to things like the selection and the input focus. Tk_CanvasGetTextInfo may be invoked by a type manager to obtain additional information needed for items that display text. The return value from Tk_CanvasGetTextInfo is a pointer to a structure that is shared between Tk and all the items that display text. The structure has the following form:

```c
typedef struct Tk_CanvasTextInfo {
    Tk_3DBorder selBorder;
    int selBorderWidth;
    XColor *selFgColorPtr;
} Tk_CanvasTextInfo;
```
Tk_Item *selItemPtr;
int selectFirst;
int selectLast;
Tk_Item *anchorItemPtr;
int selectAnchor;
Tk_3DBorder insertBorder;
int insertWidth;
int insertBorderWidth;
Tk_Item *focusItemPtr;
int gotFocus;
int cursorOn;
} Tk.CanvasTextInfo;

The selBorder field identifies a Tk_3DBorder that should be used for drawing the background under selected text. selBorderWidth gives the width of the raised border around selected text, in pixels. selFgColorPtr points to an XColor that describes the foreground color to be used when drawing selected text. selItemPtr points to the item that is currently selected, or NULL if there is no item selected or if the canvas does not have the selection. selectFirst and selectLast give the indices of the first and last selected characters in selItemPtr, as returned by the indexProc for that item. anchorItemPtr points to the item that currently has the selection anchor; this is not necessarily the same as selItemPtr. selectAnchor is an index that identifies the anchor position within anchorItemPtr. insertBorder contains a Tk_3DBorder to use when drawing the insertion cursor; insertWidth gives the total width of the insertion cursor in pixels, and insertBorderWidth gives the width of the raised border around the insertion cursor. focusItemPtr identifies the item that currently has the input focus, or NULL if there is no such item. gotFocus is 1 if the canvas widget has the input focus and 0 otherwise. cursorOn is 1 if the insertion cursor should be drawn in focusItemPtr and 0 if it should not be drawn; this field is toggled on and off by Tk to make the cursor blink.

The structure returned by Tk_CanvasGetTextInfo is shared between Tk and the type managers; typically the type manager calls
Tk_CanvasGetTextInfo once when an item is created and then saves the pointer in the item's record. Tk will update information in the Tk_CanvasTextInfo; for example, a configure widget command might change the selBorder field, or a select widget command might change the selectFirst field, or Tk might change cursorOn in order to make the insertion cursor flash on and off during successive redispays.

Type managers should treat all of the fields of the Tk_CanvasTextInfo structure as read-only, except for selItemPtr, selectFirst, selectLast, and selectAnchor. Type managers may change selectFirst, selectLast, and selectAnchor to adjust for insertions and deletions in the item (but only if the item is the current owner of the selection or anchor, as determined by selItemPtr or anchorItemPtr). If all of the selected text in the item is deleted, the item should set selItemPtr to NULL to indicate that there is no longer a selection.

**KEYWORDS**

canvas, focus, insertion cursor, selection, selection anchor, text

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
   Tk_ConfigureWidget, Tk_ConfigureInfo, Tk_ConfigureValue, Tk_FreeOptions - process configuration options for widgets

SYNOPSIS
   #include <tk.h>
   int Tk_ConfigureWidget(interp, tkwin, specs, argc, argv, widgRec, flags)
   int Tk_ConfigureInfo(interp, tkwin, specs, widgRec, argvName, flags)
   int Tk_ConfigureValue(interp, tkwin, specs, widgRec, argvName, flags)
   Tk_FreeOptions(specs, widgRec, display, flags)

ARGUMENTS

DESCRIPTION
   TK_CONFIG_ACTIVE_CURSOR
   TK_CONFIG_ANCHOR
   TK_CONFIG_BITMAP
   TK_CONFIG_BOOLEAN
   TK_CONFIG_BORDER
   TK_CONFIG_CAP_STYLE
   TK_CONFIG_COLOR
   TK_CONFIG_CURSOR
   TK_CONFIG_CUSTOM
   TK_CONFIG_DOUBLE
   TK_CONFIG_END
   TK_CONFIG_FONT
   TK_CONFIG_INT
   TK_CONFIG_JOIN_STYLE


**NAME**

Tk_ConfigureWidget, Tk_ConfigureInfo, Tk_ConfigureValue, Tk_FreeOptions - process configuration options for widgets

**SYNOPSIS**

```
#include <tk.h>
int Tk_ConfigureWidget(interp, tkwin, specs, argc, argv, widgRec, flags)
int Tk_ConfigureInfo(interp, tkwin, specs, widgRec, argvName, flags)
int Tk_ConfigureValue(interp, tkwin, specs, widgRec, argvName, flags)
```
**Tk_FreeOptions**(*specs, widgRec, display, flags*)

<table>
<thead>
<tr>
<th><strong>ARGUMENTS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp (in)</em></td>
</tr>
<tr>
<td><strong>Tk_Window</strong> tkwin (in)</td>
</tr>
<tr>
<td><strong>Tk_ConfigSpec</strong> <em>specs (in)</em></td>
</tr>
<tr>
<td>int argc (in)</td>
</tr>
<tr>
<td>const char **argv (in)</td>
</tr>
<tr>
<td>char *widgRec (in/out)</td>
</tr>
<tr>
<td>int flags (in)</td>
</tr>
</tbody>
</table>
**TK_CONFIG_USER_BIT**
and higher are used to selectively disable entries in specs.

<table>
<thead>
<tr>
<th>type name <strong>type</strong> (in)</th>
<th>The name of the type of a widget record.</th>
</tr>
</thead>
<tbody>
<tr>
<td>field name <strong>field</strong> (in)</td>
<td>The name of a field in records of type <strong>type</strong>.</td>
</tr>
<tr>
<td>const char <strong>argvName</strong> (in)</td>
<td>The name used on Tcl command lines to refer to a particular option (e.g. when creating a widget or invoking the <strong>configure</strong> widget command). If non-NULL, then information is returned only for this option. If NULL, then information is returned for all available options.</td>
</tr>
<tr>
<td>Display <strong>display</strong> (in)</td>
<td>Display containing widget whose record is being freed; needed in order to free up resources.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

Note: **Tk_ConfigureWidget** should be replaced with the new **Tcl_Obj** based API **Tk_SetOptions**. The old interface is retained for backward compatibility.

**Tk_ConfigureWidget** is called to configure various aspects of a widget, such as colors, fonts, border width, etc. It is intended as a convenience
procedure to reduce the amount of code that must be written in individual widget managers to handle configuration information. It is typically invoked when widgets are created, and again when the configure command is invoked for a widget. Although intended primarily for widgets, Tk_ConfigureWidget can be used in other situations where argc-argv information is to be used to fill in a record structure, such as configuring graphical elements for a canvas widget or entries of a menu.

Tk_ConfigureWidget processes a table specifying the configuration options that are supported (specs) and a collection of command-line arguments (argc and argv) to fill in fields of a record (widgRec). It uses the option database and defaults specified in specs to fill in fields of widgRec that are not specified in argv. Tk_ConfigureWidget normally returns the value TCL_OK; in this case it does not modify interp. If an error occurs then TCL_ERROR is returned and Tk_ConfigureWidget will leave an error message in interp->result in the standard Tcl fashion. In the event of an error return, some of the fields of widgRec could already have been set, if configuration information for them was successfully processed before the error occurred. The other fields will be set to reasonable initial values so that Tk_FreeOptions can be called for cleanup.

The specs array specifies the kinds of configuration options expected by the widget. Each of its entries specifies one configuration option and has the following structure:

typedef struct {
    int type;
    char *argvName;
    char *dbName;
    char *dbClass;
    char *defValue;
    int offset;
    int specFlags;
    Tk_CustomOption *customPtr;
} Tk_ConfigSpec;
The type field indicates what type of configuration option this is (e.g. TK_CONFIG_COLOR for a color value, or TK_CONFIG_INT for an integer value). The type field indicates how to use the value of the option (more on this below). The argvName field is a string such as “-font” or “-bg”, which is compared with the values in argv (if argvName is NULL it means this is a grouped entry; see GROUPED ENTRIES below). The dbName and dbClass fields are used to look up a value for this option in the option database. The defValue field specifies a default value for this configuration option if no value is specified in either argv or the option database. Offset indicates where in widgRec to store information about this option, and specFlags contains additional information to control the processing of this configuration option (see FLAGS below). The last field, customPtr, is only used if type is TK_CONFIGCUSTOM; see CUSTOM OPTION TYPES below.

Tk_ConfigureWidget first processes argv to see which (if any) configuration options are specified there. argv must contain an even number of fields; the first of each pair of fields must match the argvName of some entry in specs (unique abbreviations are acceptable), and the second field of the pair contains the value for that configuration option. If there are entries in spec for which there were no matching entries in argv, Tk_ConfigureWidget uses the dbName and dbClass fields of the specs entry to probe the option database; if a value is found, then it is used as the value for the option. Finally, if no entry is found in the option database, the defValue field of the specs entry is used as the value for the configuration option. If the defValue is NULL, or if the TK_CONFIG_DONT_SET_DEFAULT bit is set in flags, then there is no default value and this specs entry will be ignored if no value is specified in argv or the option database.

Once a string value has been determined for a configuration option, Tk_ConfigureWidget translates the string value into a more useful form, such as a color if type is TK_CONFIG_COLOR or an integer if type is TK_CONFIG_INT. This value is then stored in the record pointed to by widgRec. This record is assumed to contain information relevant to the manager of the widget; its exact type is unknown to
Tk_ConfigureWidget. The offset field of each specs entry indicates where in widgRec to store the information about this configuration option. You should use the Tk_Offset macro to generate offset values (see below for a description of Tk_Offset). The location indicated by widgRec and offset will be referred to as the “target” in the descriptions below.

The type field of each entry in specs determines what to do with the string value of that configuration option. The legal values for type, and the corresponding actions, are:

**TK_CONFIG_ACTIVE_CURSOR**

The value must be an ASCII string identifying a cursor in a form suitable for passing to Tk_GetCursor. The value is converted to a Tk_Cursor by calling Tk_GetCursor and the result is stored in the target. In addition, the resulting cursor is made the active cursor for tkwin by calling XDefineCursor. If TK_CONFIG_NULL_OK is specified in specFlags then the value may be an empty string, in which case the target and tkwin's active cursor will be set to None. If the previous value of the target was not None, then it is freed by passing it to Tk_FreeCursor.

**TK_CONFIG_ANCHOR**

The value must be an ASCII string identifying an anchor point in one of the ways accepted by Tk_GetAnchor. The string is converted to a Tk_Anchor by calling Tk_GetAnchor and the result is stored in the target.

**TK_CONFIG_BITMAP**

The value must be an ASCII string identifying a bitmap in a form suitable for passing to Tk_GetBitmap. The value is converted to a Pixmap by calling Tk_GetBitmap and the result is stored in the target. If TK_CONFIG_NULL_OK is specified in specFlags then the value may be an empty string, in which case the target is set to None. If the previous value of the target was not None, then it is freed by passing it to Tk_FreeBitmap.

**TK_CONFIG_BOOLEAN**
The value must be an ASCII string specifying a boolean value. Any of the values “true”, “yes”, “on”, or “1”, or an abbreviation of one of these values, means true; any of the values “false”, “no”, “off”, or “0”, or an abbreviation of one of these values, means false. The target is expected to be an integer; for true values it will be set to 1 and for false values it will be set to 0.

**TK_CONFIG_BORDER**
The value must be an ASCII string identifying a border color in a form suitable for passing to `Tk_Get3DBorder`. The value is converted to a `(Tk_3DBorder *)` by calling `Tk_Get3DBorder` and the result is stored in the target. If `TK_CONFIG_NULL_OK` is specified in `specFlags` then the value may be an empty string, in which case the target will be set to NULL. If the previous value of the target was not NULL, then it is freed by passing it to `Tk_Free3DBorder`.

**TK_CONFIG_CAP_STYLE**
The value must be an ASCII string identifying a cap style in one of the ways accepted by `Tk_GetCapStyle`. The string is converted to an integer value corresponding to the cap style by calling `Tk_GetCapStyle` and the result is stored in the target.

**TK_CONFIG_COLOR**
The value must be an ASCII string identifying a color in a form suitable for passing to `Tk_GetColor`. The value is converted to an `(XColor *)` by calling `Tk_GetColor` and the result is stored in the target. If `TK_CONFIG_NULL_OK` is specified in `specFlags` then the value may be an empty string, in which case the target will be set to `None`. If the previous value of the target was not NULL, then it is freed by passing it to `Tk_FreeColor`.

**TK_CONFIG_CURSOR**
This option is identical to `TK_CONFIG_ACTIVE_CURSOR` except that the new cursor is not made the active one for `tkwin`.

**TK_CONFIGCUSTOM**
This option allows applications to define new option types. The
customPtr field of the entry points to a structure defining the new option type. See the section **CUSTOM OPTION TYPES** below for details.

**TK_CONFIG_DOUBLE**
The value must be an ASCII floating-point number in the format accepted by `strtol`. The string is converted to a `double` value, and the value is stored in the target.

**TK_CONFIG_END**
Marks the end of the table. The last entry in specs must have this type; all of its other fields are ignored and it will never match any arguments.

**TK_CONFIG_FONT**
The value must be an ASCII string identifying a font in a form suitable for passing to `Tk_GetFont`. The value is converted to a `Tk_Font` by calling `Tk_GetFont` and the result is stored in the target. If `TK_CONFIG_NULL_OK` is specified in `specFlags` then the value may be an empty string, in which case the target will be set to NULL. If the previous value of the target was not NULL, then it is freed by passing it to `Tk_FreeFont`.

**TK_CONFIG_INT**
The value must be an ASCII integer string in the format accepted by `strtol` (e.g. “0” and “0x” prefixes may be used to specify octal or hexadecimal numbers, respectively). The string is converted to an integer value and the integer is stored in the target.

**TK_CONFIG_JOIN_STYLE**
The value must be an ASCII string identifying a join style in one of the ways accepted by `Tk_GetJoinStyle`. The string is converted to an integer value corresponding to the join style by calling `Tk_GetJoinStyle` and the result is stored in the target.

**TK_CONFIG_JUSTIFY**
The value must be an ASCII string identifying a justification method in one of the ways accepted by `Tk_GetJustify`. The string is
converted to a Tk_Justify by calling Tk_GetJustify and the result is stored in the target.

TK_CONFIG_MM
The value must specify a screen distance in one of the forms acceptable to Tk_GetScreenMM. The string is converted to double-precision floating-point distance in millimeters and the value is stored in the target.

TK_CONFIG_PIXELS
The value must specify screen units in one of the forms acceptable to Tk_GetPixels. The string is converted to an integer distance in pixels and the value is stored in the target.

TK_CONFIG_RELIEF
The value must be an ASCII string identifying a relief in a form suitable for passing to Tk_GetRelief. The value is converted to an integer relief value by calling Tk_GetRelief and the result is stored in the target.

TK_CONFIG_STRING
A copy of the value is made by allocating memory space with Tcl_Alloc and copying the value into the dynamically-allocated space. A pointer to the new string is stored in the target. If TK_CONFIG_NULL_OK is specified in specFlags then the value may be an empty string, in which case the target will be set to NULL. If the previous value of the target was not NULL, then it is freed by passing it to Tcl_Free.

TK_CONFIG_SYNONYM
This type value identifies special entries in specs that are synonyms for other entries. If an argv value matches the argvName of a TK_CONFIG_SYNONYM entry, the entry is not used directly. Instead, Tk_ConfigureWidget searches specs for another entry whose argvName is the same as the dbName field in the TK_CONFIG_SYNONYM entry; this new entry is used just as if its argvName had matched the argv value. The synonym mechanism allows multiple argv values to be used for a single configuration
option, such as “-background” and “-bg”.

**TK_CONFIG_UID**

The value is translated to a *Tk_Uid* (by passing it to *Tk_GetUid*). The resulting value is stored in the target. If *TK_CONFIG_NULL_OK* is specified in *specFlags* and the value is an empty string then the target will be set to NULL.

**TK_CONFIG_WINDOW**

The value must be a window path name. It is translated to a *Tk_Window* token and the token is stored in the target.

---

**GROUPED ENTRIES**

In some cases it is useful to generate multiple resources from a single configuration value. For example, a color name might be used both to generate the background color for a widget (using *TK_CONFIG_COLOR*) and to generate a 3-D border to draw around the widget (using *TK_CONFIG_BORDER*). In cases like this it is possible to specify that several consecutive entries in *specs* are to be treated as a group. The first entry is used to determine a value (using its *argvName*, *dbName*, *dbClass*, and *defValue* fields). The value will be processed several times (one for each entry in the group), generating multiple different resources and modifying multiple targets within *widgRec*. Each of the entries after the first must have a NULL value in its *argvName* field; this indicates that the entry is to be grouped with the entry that precedes it. Only the *type* and *offset* fields are used from these follow-on entries.

---

**FLAGS**

The *flags* argument passed to *Tk_ConfigureWidget* is used in conjunction with the *specFlags* fields in the entries of *specs* to provide additional control over the processing of configuration options. These values are used in three different ways as described below.

First, if the *flags* argument to *Tk_ConfigureWidget* has the *TK_CONFIG_ARGV_ONLY* bit set (i.e., *flags |
TK_CONFIG_ARGV_ONLY != 0), then the option database and defValue fields are not used. In this case, if an entry in specs does not match a field in argv then nothing happens: the corresponding target is not modified. This feature is useful when the goal is to modify certain configuration options while leaving others in their current state, such as when a configure widget command is being processed.

Second, the specFlags field of an entry in specs may be used to control the processing of that entry. Each specFlags field may consists of an OR-ed combination of the following values:

**TK_CONFIG_COLOR_ONLY**
If this bit is set then the entry will only be considered if the display for tkwin has more than one bit plane. If the display is monochromatic then this specs entry will be ignored.

**TK_CONFIG_MONO_ONLY**
If this bit is set then the entry will only be considered if the display for tkwin has exactly one bit plane. If the display is not monochromatic then this specs entry will be ignored.

**TK_CONFIG_NULL_OK**
This bit is only relevant for some types of entries (see the descriptions of the various entry types above). If this bit is set, it indicates that an empty string value for the field is acceptable and if it occurs then the target should be set to NULL or None, depending on the type of the target. This flag is typically used to allow a feature to be turned off entirely, e.g. set a cursor value to None so that a window simply inherits its parent's cursor. If this bit is not set then empty strings are processed as strings, which generally results in an error.

**TK_CONFIG_DONT_SET_DEFAULT**
If this bit is one, it means that the defValue field of the entry should only be used for returning the default value in Tk_ConfigureInfo. In calls to Tk_ConfigureWidget no default will be supplied for entries with this flag set; it is assumed that the caller has already supplied a default value in the target location. This flag provides a
performance optimization where it is expensive to process the
default string: the client can compute the default once, save the
value, and provide it before calling \texttt{Tk\_Configure\_Widget}.

\textbf{TK\_CONFIG\_OPTION\_SPECIFIED}
This bit is deprecated. It used to be set and cleared by
\texttt{Tk\_Configure\_Widget} so that callers could detect what entries
were specified in \texttt{argv}, but it was removed because it was
inherently thread-unsafe. Code that wishes to detect what options
were specified should use \texttt{Tk\_Set\_Options} instead.

The \texttt{TK\_CONFIG\_MONO\_ONLY} and \texttt{TK\_CONFIG\_COLOR\_ONLY}
flags are typically used to specify different default values for
monochrome and color displays. This is done by creating two entries in
\texttt{specs} that are identical except for their \texttt{defValue} and \texttt{specFlags} fields.
One entry should have the value \texttt{TK\_CONFIG\_MONO\_ONLY} in its
\texttt{specFlags} and the default value for monochrome displays in its
\texttt{defValue}; the other entry should have the value
\texttt{TK\_CONFIG\_COLOR\_ONLY} in its \texttt{specFlags} and the appropriate
\texttt{defValue} for color displays.

Third, it is possible to use \texttt{flags} and \texttt{specFlags} together to selectively
disable some entries. This feature is not needed very often. It is useful
in cases where several similar kinds of widgets are implemented in one
place. It allows a single \texttt{specs} table to be created with all the
configuration options for all the widget types. When processing a
particular widget type, only entries relevant to that type will be used.
This effect is achieved by setting the high-order bits (those in positions
equal to or greater than \texttt{TK\_CONFIG\_USER\_BIT}) in \texttt{specFlags} values
or in \texttt{flags}. In order for a particular entry in \texttt{specs} to be used, its high-
order bits must match exactly the high-order bits of the \texttt{flags} value
passed to \texttt{Tk\_Configure\_Widget}. If a \texttt{specs} table is being used for \texttt{N}
different widget types, then \texttt{N} of the high-order bits will be used. Each
\texttt{specs} entry will have one of more of those bits set in its \texttt{specFlags} field
to indicate the widget types for which this entry is valid. When calling
\texttt{Tk\_Configure\_Widget}, \texttt{flags} will have a single one of these bits set to
select the entries for the desired widget type. For a working example of
this feature, see the code in \texttt{tkButton.c}. 
**TK_OFFSET**

The `Tk_Offset` macro is provided as a safe way of generating the offset values for entries in Tk_ConfigSpec structures. It takes two arguments: the name of a type of record, and the name of a field in that record. It returns the byte offset of the named field in records of the given type.

**TK_CONFIGUREINFO**

The `Tk_ConfigureInfo` procedure may be used to obtain information about one or all of the options for a given widget. Given a token for a window (`tkwin`), a table describing the configuration options for a class of widgets (`specs`), a pointer to a widget record containing the current information for a widget (`widgRec`), and a NULL `argvName` argument, `Tk_ConfigureInfo` generates a string describing all of the configuration options for the window. The string is placed in `interp->result`. Under normal circumstances it returns `TCL_OK`; if an error occurs then it returns `TCL_ERROR` and `interp->result` contains an error message.

If `argvName` is NULL, then the value left in `interp->result` by `Tk_ConfigureInfo` consists of a list of one or more entries, each of which describes one configuration option (i.e. one entry in `specs`). Each entry in the list will contain either two or five values. If the corresponding entry in `specs` has type `TK_CONFIG_SYNONYM`, then the list will contain two values: the `argvName` for the entry and the `dbName` (synonym name). Otherwise the list will contain five values: `argvName`, `dbName`, `dbClass`, `defValue`, and current value. The current value is computed from the appropriate field of `widgRec` by calling procedures like `Tk_NameOfColor`.

If the `argvName` argument to `Tk_ConfigureInfo` is non-NULL, then it indicates a single option, and information is returned only for that option. The string placed in `interp->result` will be a list containing two or five values as described above; this will be identical to the corresponding sublist that would have been returned if `argvName` had been NULL.
The flags argument to `Tk_ConfigureInfo` is used to restrict the specs entries to consider, just as for `Tk_ConfigureWidget`.

**TK_CONFIGUREVALUE**

`Tk_ConfigureValue` takes arguments similar to `Tk_ConfigureInfo`; instead of returning a list of values, it just returns the current value of the option given by `argvName` (`argvName` must not be NULL). The value is returned in `interp->result` and `TCL_OK` is normally returned as the procedure's result. If an error occurs in `Tk_ConfigureValue` (e.g., `argvName` is not a valid option name), `TCL_ERROR` is returned and an error message is left in `interp->result`. This procedure is typically called to implement `cget` widget commands.

**TK_FREEOPTIONS**

The `Tk_FreeOptions` procedure may be invoked during widget cleanup to release all of the resources associated with configuration options. It scans through specs and for each entry corresponding to a resource that must be explicitly freed (e.g. those with type `TK_CONFIG_COLOR`), it frees the resource in the widget record. If the field in the widget record does not refer to a resource (e.g. it contains a null pointer) then no resource is freed for that entry. After freeing a resource, `Tk_FreeOptions` sets the corresponding field of the widget record to null.

**CUSTOM OPTION TYPES**

Applications can extend the built-in configuration types with additional configuration types by writing procedures to parse and print options of the a type and creating a structure pointing to those procedures:

```c
typedef struct Tk_CustomOption {
    Tk_OptionParseProc *parseProc;
    Tk_OptionPrintProc *printProc;
    ClientData clientData;
} Tk_CustomOption;
```
```c
typedef int Tk_OptionParseProc(
    ClientData clientData,
    Tcl_Interp *interp,
    Tk_Window tkwin,
    char *value,
    char *widgRec,
    int offset);

typedef char *Tk_OptionPrintProc(
    ClientData clientData,
    Tk_Window tkwin,
    char *widgRec,
    int offset,
    Tcl_FreeProc **freeProcPtr);
```

The Tk_CustomOption structure contains three fields, which are pointers to the two procedures and a `clientData` value to be passed to those procedures when they are invoked. The `clientData` value typically points to a structure containing information that is needed by the procedures when they are parsing and printing options.

The `parseProc` procedure is invoked by `Tk_ConfigureWidget` to parse a string and store the resulting value in the widget record. The `clientData` argument is a copy of the `clientData` field in the Tk_CustomOption structure. The `interp` argument points to a Tcl interpreter used for error reporting. `Tkwin` is a copy of the `tkwin` argument to `Tk_ConfigureWidget`. The `value` argument is a string describing the value for the option; it could have been specified explicitly in the call to `Tk_ConfigureWidget` or it could come from the option database or a default. `Value` will never be a null pointer but it may point to an empty string. `RecordPtr` is the same as the `widgRec` argument to `Tk_ConfigureWidget`; it points to the start of the widget record to modify. The last argument, `offset`, gives the offset in bytes from the start of the widget record to the location where the option value is to be placed. The procedure should translate the string to whatever
form is appropriate for the option and store the value in the widget record. It should normally return TCL_OK, but if an error occurs in translating the string to a value then it should return TCL_ERROR and store an error message in interp->result.

The printProc procedure is called by Tk_ConfigureInfo to produce a string value describing an existing option. Its clientData, tkwin, widgRec, and offset arguments all have the same meaning as for Tk_OptionParseProc procedures. The printProc procedure should examine the option whose value is stored at offset in widgRec, produce a string describing that option, and return a pointer to the string. If the string is stored in dynamically-allocated memory, then the procedure must set *freeProcPtr to the address of a procedure to call to free the string's memory; Tk_ConfigureInfo will call this procedure when it is finished with the string. If the result string is stored in static memory then printProc need not do anything with the freeProcPtr argument.

Once parseProc and printProc have been defined and a Tk_CustomOption structure has been created for them, options of this new type may be manipulated with Tk_ConfigSpec entries whose type fields are TK_CONFIG_CUSTOM and whose customPtr fields point to the Tk_CustomOption structure.

**EXAMPLES**

Although the explanation of Tk_ConfigureWidget is fairly complicated, its actual use is pretty straightforward. The easiest way to get started is to copy the code from an existing widget. The library implementation of frames (tkFrame.c) has a simple configuration table, and the library implementation of buttons (tkButton.c) has a much more complex table that uses many of the fancy specFlags mechanisms.

**SEE ALSO**

Tk_SetOptions

**KEYWORDS**
anchor, bitmap, boolean, border, cap style, color, configuration options, cursor, custom, double, font, integer, join style, justify, millimeters, pixels, relief, synonym, uid

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tk_InternAtom, Tk_GetAtomName - manage cache of X atoms

SYNOPSIS
#include <tk.h>
Atom
Tk_InternAtom(tkwin, name)
const char *
Tk_GetAtomName(tkwin, atom)

ARGUMENTS
DESCRIPTION
KEYWORDS
**DESCRIPTION**

These procedures are similar to the Xlib procedures `XInternAtom` and `XGetAtomName`. `Tk_InternAtom` returns the atom identifier associated with string given by `name`; the atom identifier is only valid for the display associated with `tkwin`. `Tk_GetAtomName` returns the string associated with `atom` on `tkwin`'s display. The string returned by `Tk_GetAtomName` is in Tk's storage: the caller need not free this space when finished with the string, and the caller should not modify the contents of the returned string. If there is no atom `atom` on `tkwin`'s display, then `Tk_GetAtomName` returns the string “?bad atom?”.

Tk caches the information returned by `Tk_InternAtom` and `Tk_GetAtomName` so that future calls for the same information can be serviced from the cache without contacting the server. Thus `Tk_InternAtom` and `Tk_GetAtomName` are generally much faster than their Xlib counterparts, and they should be used in place of the Xlib procedures.

**KEYWORDS**

atom, cache, display
NAME

Tk_GetPixmap, Tk_FreePixmap - allocate and free pixmaps

SYNOPSIS

#include <tk.h>
Pixmap
Tk_GetPixmap(display, d, width, height, depth)
Tk_FreePixmap(display, pixmap)

ARGUMENTS

Display *display (in) X display for the pixmap.

Drawable d (in) Pixmap or window where the new pixmap will be used for drawing.

int width (in) Width of pixmap.

int height (in) Height of pixmap.

int depth (in) Number of bits per pixel in pixmap.

Pixmap pixmap (in) Pixmap to destroy.

DESCRIPTION

These procedures are identical to the Xlib procedures XCreatePixmap
and \texttt{XFreePixmap}, except that they have extra code to manage X resource identifiers so that identifiers for deleted pixmaps can be reused in the future. It is important for Tk applications to use these procedures rather than \texttt{XCreatePixmap} and \texttt{XFreePixmap}; otherwise long-running applications may run out of resource identifiers.

\texttt{Tk\_GetPixmap} creates a pixmap suitable for drawing in \textit{d}, with dimensions given by \textit{width}, \textit{height}, and \textit{depth}, and returns its identifier. \texttt{Tk\_FreePixmap} destroys the pixmap given by \textit{pixmap} and makes its resource identifier available for reuse.

**KEYWORDS**

\texttt{pixmap}, \texttt{resource identifier}

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_FreeXId - make X resource identifier available for reuse

SYNOPSIS

```
#include <tk.h>
Tk_FreeXId(display, id)
```

ARGUMENTS

<table>
<thead>
<tr>
<th>Display *display (in)</th>
<th>Display for which id was allocated.</th>
</tr>
</thead>
<tbody>
<tr>
<td>XID id (in)</td>
<td>Identifier of X resource (window, font, pixmap, cursor, graphics context, or colormap) that is no longer in use.</td>
</tr>
</tbody>
</table>

DESCRIPTION

The default allocator for resource identifiers provided by Xlib is very simple-minded and does not allow resource identifiers to be re-used. If a long-running application reaches the end of the resource id space, it will generate an X protocol error and crash. Tk replaces the default id allocator with its own allocator, which allows identifiers to be reused. In order for this to work, **Tk_FreeXId** must be called to tell the allocator about resources that have been freed. Tk automatically calls **Tk_FreeXId** whenever it frees a resource, so if you use procedures like **Tk_GetFont**, **Tk_GetGC**, and **Tk_GetPixmap** then you need not call...
Tk_FreeXId. However, if you allocate resources directly from Xlib, for example by calling XCreatePixmap, then you should call Tk_FreeXId when you call the corresponding Xlib free procedure, such as XFreePixmap. If you do not call Tk_FreeXId then the resource identifier will be lost, which could cause problems if the application runs long enough to lose all of the available identifiers.

**KEYWORDS**

resource identifier
NAME

Tk_ClearSelection - Deselect a selection

SYNOPSIS

#include <tk.h>
Tk_ClearSelection(tkwin, selection)

ARGUMENTS

<table>
<thead>
<tr>
<th>Tk_Window tkwin (in)</th>
<th>The selection will be cleared from the display containing this window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atom selection (in)</td>
<td>The name of selection to be cleared.</td>
</tr>
</tbody>
</table>

DESCRIPTION

Tk_ClearSelection cancels the selection specified by the atom selection for the display containing tkwin. The selection need not be in tkwin itself or even in tkwin's application. If there is a window anywhere on tkwin's display that owns selection, the window will be notified and the selection will be cleared. If there is no owner for selection on the display, then the procedure has no effect.

KEYWORDS

clear, selection
NAME

Tk_SetWindowVisual - change visual characteristics of window

SYNOPSIS

```
#include <tk.h>
int
Tk_SetWindowVisual(tkwin, visual, depth, colormap)
```

ARGUMENTS

<table>
<thead>
<tr>
<th>Tk_Winow tkwin (in)</th>
<th>Token for window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visual *visual (in)</td>
<td>New visual type to use for tkwin.</td>
</tr>
<tr>
<td>int depth (in)</td>
<td>Number of bits per pixel desired for tkwin.</td>
</tr>
<tr>
<td>Colormap colormap (in)</td>
<td>New colormap for tkwin, which must be compatible with visual and depth.</td>
</tr>
</tbody>
</table>

DESCRIPTION

When Tk creates a new window it assigns it the default visual characteristics (visual, depth, and colormap) for its screen. Tk_SetWindowVisual may be called to change them. Tk_SetWindowVisual must be called before the window has actually been created in X (e.g. before Tk_MapWindow or
Tk_MakeWindowExist has been invoked for the window). The safest thing is to call Tk_SetWindowVisual immediately after calling Tk_CreateWindow. If tkwin has already been created before Tk_SetWindowVisual is called then it returns 0 and does not make any changes; otherwise it returns 1 to signify that the operation completed successfully.

Note: Tk_SetWindowVisual should not be called if you just want to change a window's colormap without changing its visual or depth; call Tk_SetWindowColormap instead.

KEYWORDS
colormap, depth, visual
NAME
Tk_ClipboardClear, Tk_ClipboardAppend - Manage the clipboard

SYNOPSIS
#include <tk.h>

int Tk_ClipboardClear(interp, tkwin)

int Tk_ClipboardAppend(interp, tkwin, target, format, buffer)

ARGUMENTS

Tcl_Interp *interp (in)  Interpreter to use for reporting errors.

Tk_Window tkwin (in)  Window that determines which display's clipboard
DESCRIPTION

These two procedures manage the clipboard for Tk. The clipboard is typically managed by calling `Tk_ClipboardClear` once, then calling `Tk_ClipboardAppend` to add data for any number of targets.

`Tk_ClipboardClear` claims the CLIPBOARD selection and frees any data items previously stored on the clipboard in this application. It normally returns `TCL_OK`, but if an error occurs it returns `TCL_ERROR` and leaves an error message in `interp->result`. `Tk_ClipboardClear` must be called before a sequence of `Tk_ClipboardAppend` calls can be issued.

`Tk_ClipboardAppend` appends a buffer of data to the clipboard. The first buffer for a given `target` determines the `format` for that `target`. Any successive appends for that `target` must have the same format or an error will be returned. `Tk_ClipboardAppend` returns `TCL_OK` if the buffer is successfully copied onto the clipboard. If the clipboard is not currently owned by the application, either because `Tk_ClipboardClear`
has not been called or because ownership of the clipboard has changed since the last call to Tk_ClipboardClear, Tk_ClipboardAppend returns TCL_ERROR and leaves an error message in interp->result.

In order to guarantee atomicity, no event handling should occur between Tk_ClipboardClear and the following Tk_ClipboardAppend calls (otherwise someone could retrieve a partially completed clipboard or claim ownership away from this application).

Tk_ClipboardClear may invoke callbacks, including arbitrary Tcl scripts, as a result of losing the CLIPBOARD selection, so any calling function should take care to be reentrant at the point Tk_ClipboardClear is invoked.

**KEYWORDS**

append, clipboard, clear, format, type
NAME

Tk_Main - main program for Tk-based applications

SYNOPSIS

#include <tk.h>
Tk_Main(argc, argv, appInitProc)

ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int argc (in)</td>
<td>Number of elements in argv.</td>
</tr>
<tr>
<td>char *argv[] (in)</td>
<td>Array of strings containing command-line arguments.</td>
</tr>
<tr>
<td>Tcl_AppInitProc *appInitProc (in)</td>
<td>Address of an application-specific initialization procedure. The value for this argument is usually Tcl_AppInit.</td>
</tr>
</tbody>
</table>

DESCRIPTION

Tk_Main acts as the main program for most Tk-based applications. Starting with Tk 4.0 it is not called main anymore because it is part of the Tk library and having a function main in a library (particularly a shared library) causes problems on many systems. Having main in the Tk library would also make it hard to use Tk in C++ programs, since C++ programs must have special C++ main functions.
Normally each application contains a small `main` function that does nothing but invoke `Tk_Main`. `Tk_Main` then does all the work of creating and running a `wish`-like application.

When it is has finished its own initialization, but before it processes commands, `Tk_Main` calls the procedure given by the `appInitProc` argument. This procedure provides a “hook” for the application to perform its own initialization, such as defining application-specific commands. The procedure must have an interface that matches the type `Tcl_AppInitProc`:

```c
typedef int Tcl_AppInitProc(Tcl_Interp *interp);
```

`AppInitProc` is almost always a pointer to `Tcl_AppInit`; for more details on this procedure, see the documentation for `Tcl_AppInit`.

**KEYWORDS**

application-specific initialization, command-line arguments, main program

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_MainLoop - loop for events until all windows are deleted

SYNOPSIS

#include <tk.h>
Tk_MainLoop()

DESCRIPTION

Tk_MainLoop is a procedure that loops repeatedly calling Tcl_DoOneEvent. It returns only when there are no applications left in this process (i.e. no main windows exist anymore). Most windowing applications will call Tk_MainLoop after initialization; the main execution of the application will consist entirely of callbacks invoked via Tcl_DoOneEvent.

KEYWORDS

application, event, main loop

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1992 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tk_GetAnchorFromObj, Tk_GetAnchor, Tk_NameOfAnchor - translate between strings and anchor positions

SYNOPSIS
#include <tk.h>
int
Tk_GetAnchorFromObj(interp, objPtr, anchorPtr)
int
Tk_GetAnchor(interp, string, anchorPtr)
const char *
Tk_NameOfAnchor(anchor)

ARGUMENTS
DESCRIPTION
KEYWORDS

NAME
Tk_GetAnchorFromObj, Tk_GetAnchor, Tk_NameOfAnchor - translate between strings and anchor positions

SYNOPSIS
#include <tk.h>
int
Tk_GetAnchorFromObj(interp, objPtr, anchorPtr)
int
Tk_GetAnchor(interp, string, anchorPtr)
const char *
Tk_NameOfAnchor(anchor)

ARGUMENTS
<table>
<thead>
<tr>
<th><strong>Tcl_Interp</strong> <em>interp</em> (in)</th>
<th>Interpreter to use for error reporting, or NULL.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Obj</strong> <em>objPtr</em> (in/out)</td>
<td>String value contains name of anchor point: “n”, “ne”, “e”, “se”, “s”, “sw”, “w”, “nw”, or “center”; internal rep will be modified to cache corresponding Tk_Anchor.</td>
</tr>
<tr>
<td><strong>const char</strong> <em>string</em> (in)</td>
<td>Same as <em>objPtr</em> except description of anchor point is passed as a string.</td>
</tr>
<tr>
<td><strong>int</strong> <em>anchorPtr</em> (out)</td>
<td>Pointer to location in which to store anchor position corresponding to <em>objPtr</em> or <em>string</em>.</td>
</tr>
<tr>
<td><strong>Tk_Anchor</strong> anchor (in)</td>
<td>Anchor position, e.g. <strong>TCL_ANCHOR_CENTER</strong>.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

**Tk_GetAnchorFromObj** places in *anchorPtr* an anchor position (enumerated type **Tk_Anchor**) corresponding to *objPtr*’s value. The result will be one of **TK_ANCHOR_N**, **TK_ANCHOR_NE**, **TK_ANCHOR_E**, **TK_ANCHOR_SE**, **TK_ANCHOR_S**, **TK_ANCHOR_SW**, **TK_ANCHOR_W**, **TK_ANCHOR_NW**, or **TK_ANCHOR_CENTER**. Anchor positions are typically used for indicating a point on an object that will be used to position the object, e.g. **TK_ANCHOR_N** means position the top center point of the object at a particular place.

Under normal circumstances the return value is **TCL_OK** and *interp* is
unused. If string does not contain a valid anchor position or an abbreviation of one of these names, TCL_ERROR is returned, *anchorPtr is unmodified, and an error message is stored in interp's result if interp is not NULL. Tk_GetAnchorFromObj caches information about the return value in objPtr, which speeds up future calls to Tk_GetAnchorFromObj with the same objPtr.

Tk_GetAnchor is identical to Tk_GetAnchorFromObj except that the description of the anchor is specified with a string instead of an object. This prevents Tk_GetAnchor from caching the return value, so Tk_GetAnchor is less efficient than Tk_GetAnchorFromObj.

Tk_NameOfAnchor is the logical inverse of Tk_GetAnchor. Given an anchor position such as TK_ANCHOR_N it returns a statically-allocated string corresponding to anchor. If anchor is not a legal anchor value, then “unknown anchor position” is returned.

KEYWORDS

anchor position

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1998 Sun Microsystems, Inc.
**NAME**
Tk_MaintainGeometry, Tk_UnmaintainGeometry - maintain geometry of one window relative to another

**SYNOPSIS**
```
#include <tk.h>
Tk_MaintainGeometry(slave, master, x, y, width, height)
Tk_UnmaintainGeometry(slave, master)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Window slave (in)</td>
<td>Window whose geometry is to be controlled.</td>
</tr>
<tr>
<td>Tk_Window master (in)</td>
<td>Window relative to which slave's geometry will be controlled.</td>
</tr>
<tr>
<td>int x (in)</td>
<td>Desired x-coordinate of the slave window.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

The `Tk_MaintainGeometry` and `Tk_UnmaintainGeometry` commands allow you to control the geometry of one window relative to another. When `Tk_MaintainGeometry` is called, the geometry of the `slave` window will be maintained relative to the `master` window. The `Tk_UnmaintainGeometry` command resumes the normal resizing of the `slave` window. The `x` and `y` arguments specify the relative x and y positions, and `width` and `height` specify the relative sizes of the windows.

**KEYWORDS**

Tk, Maintain Geometry, Unmaintain Geometry, Tk Window, Relative Position, Relative Size.
slave in master, measured in pixels from the inside of master's left border to the outside of slave's left border.

<table>
<thead>
<tr>
<th>int y (in)</th>
<th>Desired y-coordinate of slave in master, measured in pixels from the inside of master's top border to the outside of slave's top border.</th>
</tr>
</thead>
<tbody>
<tr>
<td>int width (in)</td>
<td>Desired width for slave, in pixels.</td>
</tr>
<tr>
<td>int height (in)</td>
<td>Desired height for slave, in pixels.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

Tk_MaintainGeometry and Tk_UnmaintainGeometry make it easier for geometry managers to deal with slaves whose masters are not their parents. Three problems arise if the master for a slave is not its parent:

[1]

The x- and y-position of the slave must be translated from the coordinate system of the master to that of the parent before positioning the slave.

[2]

If the master window, or any of its ancestors up to the slave's parent, is moved, then the slave must be repositioned within its parent in order to maintain the correct position relative to the master.

[3]
If the master or one of its ancestors is mapped or unmapped, then the slave must be mapped or unmapped to correspond.

None of these problems is an issue if the parent and master are the same. For example, if the master or one of its ancestors is unmapped, the slave is automatically removed by the screen by X.

**Tk_MaintainGeometry** deals with these problems for slaves whose masters are not their parents, as well as handling the simpler case of slaves whose masters are their parents. **Tk_MaintainGeometry** is typically called by a window manager once it has decided where a slave should be positioned relative to its master. **Tk_MaintainGeometry** translates the coordinates to the coordinate system of slave’s parent and then moves and resizes the slave appropriately. Furthermore, it remembers the desired position and creates event handlers to monitor the master and all of its ancestors up to (but not including) the slave's parent. If any of these windows is moved, mapped, or unmapped, the slave will be adjusted so that it is mapped only when the master is mapped and its geometry relative to the master remains as specified by $x$, $y$, width, and height.

When a window manager relinquishes control over a window, or if it decides that it does not want the window to appear on the screen under any conditions, it calls **Tk_UnmaintainGeometry**. **Tk_UnmaintainGeometry** unmaps the window and cancels any previous calls to **Tk_MaintainGeometry** for the master-slave pair, so that the slave's geometry and mapped state are no longer maintained automatically. **Tk_UnmaintainGeometry** need not be called by a geometry manager if the slave, the master, or any of the master's ancestors is destroyed: Tk will call it automatically.

If **Tk_MaintainGeometry** is called repeatedly for the same master-slave pair, the information from the most recent call supersedes any older information. If **Tk_UnmaintainGeometry** is called for a master-slave pair that is not currently managed, the call has no effect.

**KEYWORDS**
geometry manager, map, master, parent, position, slave, unmap
NAME

Tk_StrictMotif - Return value of tk_strictMotif variable

SYNOPSIS

#include <tk.h>
int
Tk_StrictMotif(tkwin)

ARGUMENTS

Tk_Window tkwin (in) Token for window.

DESCRIPTION

This procedure returns the current value of the tk_strictMotif variable in the interpreter associated with tkwin's application. The value is returned as an integer that is either 0 or 1. 1 means that strict Motif compliance has been requested, so anything that is not part of the Motif specification should be avoided. 0 means that “Motif-like” is good enough, and extra features are welcome.

This procedure uses a link to the Tcl variable to provide much faster access to the variable's value than could be had by calling Tcl_GetVar.

KEYWORDS

Motif compliance, tk_strictMotif variable
NAME

Tk_MainWindow, Tk_GetNumMainWindows - functions for querying main window information

SYNOPSIS

#include <tk.h>

Tk_Window

Tk_MainWindow(interp)

int

Tk_GetNumMainWindows()

ARGUMENTS

Tk_Interp *interp (in/out)  Interpreter associated with the application.

DESCRIPTION

A main window is a special kind of toplevel window used as the outermost window in an application.

If interp is associated with a Tk application then Tk_MainWindow returns the application's main window. If there is no Tk application associated with interp then Tk_MainWindow returns NULL and leaves an error message in interp->result.

Tk_GetNumMainWindows returns a count of the number of main windows currently open in the process.
KEYWORDS

application, main window

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tk_CreateWindow, Tk_CreateWindowFromPath, Tk_DestroyWindow, Tk_MakeWindowExist - create or delete window

SYNOPSIS
#include <tk.h>
Tk_Window
Tk_CreateWindow(interp, parent, name, topLevScreen)
Tk_Window
Tk_CreateAnonymousWindow(interp, parent, topLevScreen)
Tk_Window
Tk_CreateWindowFromPath(interp, tkwin, pathName, topLevScreen)
Tk_DestroyWindow(tkwin)
Tk_MakeWindowExist(tkwin)

ARGUMENTS
DESCRIPTION
KEYWORDS
**Tk_CreateWindowFromPath**(*interp*, *tkwin*, *pathName*, *topLevScreen*)

**Tk_DestroyWindow**(*tkwin*)

**Tk_MakeWindowExist**(*tkwin*)

<table>
<thead>
<tr>
<th><strong>ARGUMENTS</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tcl_Interp</strong> <em>interp</em> (out)</td>
</tr>
<tr>
<td><strong>Tk_Window</strong> parent (in)</td>
</tr>
<tr>
<td>const char <em>name</em> (in)</td>
</tr>
<tr>
<td>const char <em>topLevScreen</em> (in)</td>
</tr>
<tr>
<td><strong>Tk_Window</strong> tkwin (in)</td>
</tr>
<tr>
<td>const char <em>pathName</em> (in)</td>
</tr>
</tbody>
</table>
within application (e.g. .a.b.c).

**DESCRIPTION**

The procedures **Tk_CreateWindow**, **Tk_CreateAnonymousWindow**, and **Tk_CreateWindowFromPath** are used to create new windows for use in Tk-based applications. Each of the procedures returns a token that can be used to manipulate the window in other calls to the Tk library. If the window could not be created successfully, then NULL is returned and *interp->result* is modified to hold an error message.

Tk supports two different kinds of windows: internal windows and top-level windows. An internal window is an interior window of a Tk application, such as a scrollbar or menu bar or button. A top-level window is one that is created as a child of a screen's root window, rather than as an interior window, but which is logically part of some existing main window. Examples of top-level windows are pop-up menus and dialog boxes.

New windows may be created by calling **Tk_CreateWindow**. If the *topLevScreen* argument is NULL, then the new window will be an internal window. If *topLevScreen* is non-NULL, then the new window will be a top-level window: *topLevScreen* indicates the name of a screen and the new window will be created as a child of the root window of *topLevScreen*. In either case Tk will consider the new window to be the logical child of *parent*: the new window's path name will reflect this fact, options may be specified for the new window under this assumption, and so on. The only difference is that new X window for a top-level window will not be a child of *parent*'s X window. For example, a pull-down menu's *parent* would be the button-like window used to invoke it, which would in turn be a child of the menu bar window. A dialog box might have the application's main window as its parent.

**Tk_CreateAnonymousWindow** differs from **Tk_CreateWindow** in that it creates an unnamed window. This window will be manipulable only using C interfaces, and will not be visible to Tcl scripts. Both interior
windows and top-level windows may be created with
**Tk_CreateAnonymousWindow**.

**Tk_CreateWindowFromPath** offers an alternate way of specifying new windows. In **Tk_CreateWindowFromPath** the new window is specified with a token for any window in the target application (*tkwin*), plus a path name for the new window. It produces the same effect as **Tk_CreateWindow** and allows both top-level and internal windows to be created, depending on the value of *topLevScreen*. In calls to **Tk_CreateWindowFromPath**, as in calls to **Tk_CreateWindow**, the parent of the new window must exist at the time of the call, but the new window must not already exist.

The window creation procedures do not actually issue the command to X to create a window. Instead, they create a local data structure associated with the window and defer the creation of the X window. The window will actually be created by the first call to **Tk_MapWindow**. Deferred window creation allows various aspects of the window (such as its size, background color, etc.) to be modified after its creation without incurring any overhead in the X server. When the window is finally mapped all of the window attributes can be set while creating the window.

The value returned by a window-creation procedure is not the X token for the window (it cannot be, since X has not been asked to create the window yet). Instead, it is a token for Tk's local data structure for the window. Most of the Tk library procedures take Tk_Window tokens, rather than X identifiers. The actual X window identifier can be retrieved from the local data structure using the **Tk_WindowId** macro; see the manual entry for **Tk_WindowId** for details.

**Tk_DestroyWindow** deletes a window and all the data structures associated with it, including any event handlers created with **Tk_CreateEventHandler**. In addition, **Tk_DestroyWindow** will delete any children of *tkwin* recursively (where children are defined in the Tk sense, consisting of all windows that were created with the given window as *parent*). If *tkwin* is an internal window, then event handlers interested in destroy events are invoked immediately. If *tkwin* is a top-
level or main window, then the event handlers will be invoked later, after X has seen the request and returned an event for it.

If a window has been created but has not been mapped, so no X window exists, it is possible to force the creation of the X window by calling **Tk_MakeWindowExist**. This procedure issues the X commands to instantiate the window given by *tkwin*.

**KEYWORDS**

create, deferred creation, destroy, display, internal window, screen, top-level window, window

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
**NAME**

Tk_ManageGeometry - arrange to handle geometry requests for a window

**SYNOPSIS**

```c
#include <tk.h>

Tk_ManageGeometry(tkwin, mgrPtr, clientData)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Window tkwin</td>
<td>Token for window to be managed.</td>
</tr>
<tr>
<td>const Tk_GeomMgr *mgrPtr</td>
<td>Pointer to data structure containing information about the geometry manager, or NULL to indicate that tkwin's geometry should not be managed anymore. The data structure pointed to by mgrPtr must be static: Tk keeps a reference to it as long as the window is managed.</td>
</tr>
<tr>
<td>ClientData clientData</td>
<td>Arbitrary one-word value to pass to geometry manager callbacks.</td>
</tr>
</tbody>
</table>
**DESCRIPTION**

**Tk_ManageGeometry** arranges for a particular geometry manager, described by the `mgrPtr` argument, to control the geometry of a particular slave window, given by `tkwin`. If `tkwin` was previously managed by some other geometry manager, the previous manager loses control in favor of the new one. If `mgrPtr` is NULL, geometry management is cancelled for `tkwin`.

The structure pointed to by `mgrPtr` contains information about the geometry manager:

```c
typedef struct {
    const char *name;
    Tk_GeomRequestProc *requestProc;
    Tk_GeomLostSlaveProc *lostSlaveProc;
} Tk_GeomMgr;
```

The `name` field is the textual name for the geometry manager, such as `pack` or `place`; this value will be returned by the command `wininfo manager`.

`requestProc` is a procedure in the geometry manager that will be invoked whenever **Tk_GeometryRequest** is called by the slave to change its desired geometry. `requestProc` should have arguments and results that match the type **Tk_GeomRequestProc**:

```c
typedef void Tk_GeomRequestProc(
    ClientData clientData,
    Tk_Window tkwin);
```

The parameters to `requestProc` will be identical to the corresponding parameters passed to **Tk_ManageGeometry**. `clientData` usually points to a data structure containing application-specific information about how to manage `tkwin`'s geometry.
The *lostSlaveProc* field of *mgrPtr* points to another procedure in the geometry manager. Tk will invoke *lostSlaveProc* if some other manager calls **Tk_ManageGeometry** to claim *tkwin* away from the current geometry manager. *lostSlaveProc* is not invoked if **Tk_ManageGeometry** is called with a NULL value for *mgrPtr* (presumably the current geometry manager has made this call, so it already knows that the window is no longer managed), nor is it called if *mgrPtr* is the same as the window's current geometry manager. *lostSlaveProc* should have arguments and results that match the following prototype:

```c
typedef void Tk_GeomLostSlaveProc(
    ClientData clientData,
    Tk_Window tkwin);
```

The parameters to *lostSlaveProc* will be identical to the corresponding parameters passed to **Tk_ManageGeometry**.

**KEYWORDS**

callback, geometry, managed, request, unmanaged

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_MapWindow, Tk_UnmapWindow - map or unmap a window

SYNOPSIS

#include <tk.h>
Tk_Window
Tk_MapWindow(tkwin)
Tk_UnmapWindow(tkwin)

ARGUMENTS

Tk_Window tkwin (in) Token for window.

DESCRIPTION

These procedures may be used to map and unmap windows managed by Tk. Tk_MapWindow maps the window given by tkwin, and also creates an X window corresponding to tkwin if it does not already exist. See the Tk_CreateWindow manual entry for information on deferred window creation. Tk_UnmapWindow unmaps tkwin’s window from the screen.

If tkwin is a child window (i.e. Tk_CreateWindow was used to create a child window), then event handlers interested in map and unmap events are invoked immediately. If tkwin is not an internal window, then the event handlers will be invoked later, after X has seen the request and returned an event for it.

These procedures should be used in place of the X procedures
XMapWindow and XUnmapWindow, since they update Tk's local data structure for tkwin. Applications using Tk should not invoke XMapWindow and XUnmapWindow directly.

KEYWORDS

map, unmap, window
NAME
Tk_GetCapStyle, Tk_NameOfCapStyle - translate between strings and cap styles

SYNOPSIS
#include <tk.h>
int
Tk_GetCapStyle(interp, string, capPtr)
const char *
Tk_NameOfCapStyle(cap)

ARGUMENTS

DESCRIPTION

KEYWORDS

NAME
Tk_GetCapStyle, Tk_NameOfCapStyle - translate between strings and cap styles

SYNOPSIS
#include <tk.h>
int
Tk_GetCapStyle(interp, string, capPtr)
const char *
Tk_NameOfCapStyle(cap)

ARGUMENTS

Tcl_Interp *interp (in)   Interpreter to use for error reporting.

const char *string (in)  String containing name of
cap style: one of “butt”, “projecting”, or “round”.

<table>
<thead>
<tr>
<th>int *capPtr (out)</th>
<th>Pointer to location in which to store X cap style corresponding to string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>int cap (in)</td>
<td>Cap style: one of CapButt, CapProjecting, or CapRound.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

`Tk_GetCapStyle` places in *capPtr the X cap style corresponding to string. This will be one of the values CapButt, CapProjecting, or CapRound. Cap styles are typically used in X graphics contexts to indicate how the end-points of lines should be capped. See the X documentation for information on what each style implies.

Under normal circumstances the return value is TCL_OK and interp is unused. If string does not contain a valid cap style or an abbreviation of one of these names, then an error message is stored in interp->result, TCL_ERROR is returned, and *capPtr is unmodified.

`Tk_NameOfCapStyle` is the logical inverse of `Tk_GetCapStyle`. Given a cap style such as CapButt it returns a statically-allocated string corresponding to cap. If cap is not a legal cap style, then “unknown cap style” is returned.

**KEYWORDS**

but, cap style, projecting, round

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_CoordsToWindow - Find window containing a point

SYNOPSIS

#include <tk.h>
Tk_Window
Tk_CoordsToWindow(rootX, rootY, tkwin)

ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int rootX</td>
<td>X-coordinate (in root window coordinates).</td>
</tr>
<tr>
<td>int rootY</td>
<td>Y-coordinate (in root window coordinates).</td>
</tr>
<tr>
<td>Tk_Window</td>
<td>Token for window that identifies application.</td>
</tr>
</tbody>
</table>

DESCRIPTION

Tk_CoordsToWindow locates the window that contains a given point. The point is specified in root coordinates with rootX and rootY (if a virtual-root window manager is in use then rootX and rootY are in the coordinate system of the virtual root window). The return value from the procedure is a token for the window that contains the given point. If the point is not in any window, or if the containing window is not in the same application as tkwin, then NULL is returned.
The containing window is decided using the same rules that determine which window contains the mouse cursor: if a parent and a child both contain the point then the child gets preference, and if two siblings both contain the point then the highest one in the stacking order (i.e. the one that’s visible on the screen) gets preference.

**KEYWORDS**

containing, coordinates, root window
NAME

Tk_MoveToplevelWindow - Adjust the position of a top-level window

SYNOPSIS

#include <tk.h>
Tk_MoveToplevelWindow(tkwin, x, y)

ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Window tkwin (in)</td>
<td>Token for top-level window to move.</td>
</tr>
<tr>
<td>int x (in)</td>
<td>New x-coordinate for the top-left pixel of tkwin's border, or the top-left pixel of the decorative border supplied for tkwin by the window manager, if there is one.</td>
</tr>
<tr>
<td>int y (in)</td>
<td>New y-coordinate for the top-left pixel of tkwin's border, or the top-left pixel of the decorative border supplied for tkwin by the window manager, if there is one.</td>
</tr>
</tbody>
</table>

DESCRIPTION
In general, a window should never set its own position; this should be done only by the geometry manager that is responsible for the window. For top-level windows the window manager is effectively the geometry manager; Tk provides interface code between the application and the window manager to convey the application’s desires to the geometry manager. The desired size for a top-level window is conveyed using the usual \texttt{Tk\_GeometryRequest} mechanism. The procedure \texttt{Tk\_MoveToplevelWindow} may be used by an application to request a particular position for a top-level window; this procedure is similar in function to the \texttt{wm\ geometry} Tcl command except that negative offsets cannot be specified. It is invoked by widgets such as menus that want to appear at a particular place on the screen.

When \texttt{Tk\_MoveToplevelWindow} is called it does not immediately pass on the new desired location to the window manager; it defers this action until all other outstanding work has been completed, using the \texttt{Tk\_DoWhenIdle} mechanism.

**KEYWORDS**

position, top-level window, window manager

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1993 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_CreateClientMessageHandler, Tk_DeleteClientMessageHandler - associate procedure callback with ClientMessage type X events

SYNOPSIS

#include <tk.h>
Tk_CreateClientMessageHandler(proc)
Tk_DeleteClientMessageHandler(proc)

ARGUMENTS

Tk_ClientMessageProc *proc (in) Procedure to invoke whenever a ClientMessage X event occurs on any display.

DESCRIPTION

Tk_CreateClientMessageHandler arranges for proc to be invoked in the future whenever a ClientMessage X event occurs that is not handled by WM_PROTOCOL. Tk_CreateClientMessageHandler is intended for use by applications which need to watch X ClientMessage events, such as drag and drop applications.

The callback to proc will be made by Tk_HandleEvent; this mechanism only works in programs that dispatch events through Tk_HandleEvent (or through other Tk procedures that call Tk_HandleEvent, such as Tk_DoOneEvent or Tk_MainLoop).
Proc should have arguments and result that match the type Tk_ClientMessageProc:

```c
typedef int Tk_ClientMessageProc(
    Tk_Window tkwin,
    XEvent *eventPtr);
```

The tkwin parameter to proc is the Tk window which is associated with this event. EventPtr is a pointer to the X event.

Whenever an X ClientMessage event is processed by Tk_HandleEvent, the proc is called if it was not handled as a WM_PROTOCOL. The return value from proc is normally 0. A non-zero return value indicates that the event is not to be handled further; that is, proc has done all processing that is to be allowed for the event.

If there are multiple ClientMessage event handlers, each one is called for each event, in the order in which they were established.

Tk_DeleteClientMessageHandler may be called to delete a previously-created ClientMessage event handler: it deletes each handler it finds that matches the proc argument. If no such handler exists, then Tk_DeleteClientMessageHandler returns without doing anything. Although Tk supports it, it's probably a bad idea to have more than one callback with the same proc argument.

KEYWORDS

bind, callback, event, handler

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 2000 Ajuba Solutions.
Tcl/Tk Applications | Tcl Commands | Tk Commands | Tcl Library | Tk Library

NAME
Tk_CreateErrorHandler, Tk_DeleteErrorHandler - handle X protocol errors

SYNOPSIS
#include <tk.h>
Tk_ErrorHandler

Tk_CreateErrorHandler(display, error, request, minor, proc, clientData)
Tk_DeleteErrorHandler(handler)

ARGUMENTS

Description

KEYWORDS

NAME
Tk_CreateErrorHandler, Tk_DeleteErrorHandler - handle X protocol errors

SYNOPSIS
#include <tk.h>
Tk_ErrorHandler

Tk_CreateErrorHandler(display, error, request, minor, proc, clientData)
Tk_DeleteErrorHandler(handler)

ARGUMENTS

Display *display (in) Display whose errors are to be handled.

int error (in) Match only error events with this value in the
error_code field. If -1, then match any error_code value.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int request (in)</td>
<td>Match only error events with this value in the request_code field. If -1, then match any request_code value.</td>
</tr>
<tr>
<td>int minor (in)</td>
<td>Match only error events with this value in the minor_code field. If -1, then match any minor_code value.</td>
</tr>
<tr>
<td>Tk_ErrorProc *proc (in)</td>
<td>Procedure to invoke whenever an error event is received for display and matches error, request, and minor. NULL means ignore any matching errors.</td>
</tr>
<tr>
<td>ClientData clientData (in)</td>
<td>Arbitrary one-word value to pass to proc.</td>
</tr>
<tr>
<td>Tk_ErrorHandler handler (in)</td>
<td>Token for error handler to delete (return value from a previous call to Tk_CreateErrorHandler).</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

Tk_CreateErrorHandler arranges for a particular procedure (proc) to be called whenever certain protocol errors occur on a particular display.
Protocol errors occur when the X protocol is used incorrectly, such as attempting to map a window that does not exist. See the Xlib documentation for `XSetErrorHandler` for more information on the kinds of errors that can occur. For `proc` to be invoked to handle a particular error, five things must occur:

1. The error must pertain to `display`.

2. Either the `error` argument to `Tk_CreateErrorHandler` must have been -1, or the `error` argument must match the `error_code` field from the error event.

3. Either the `request` argument to `Tk_CreateErrorHandler` must have been -1, or the `request` argument must match the `request_code` field from the error event.

4. Either the `minor` argument to `Tk_CreateErrorHandler` must have been -1, or the `minor` argument must match the `minor_code` field from the error event.

5. The protocol request to which the error pertains must have been made when the handler was active (see below for more information).

`Proc` should have arguments and result that match the following type:

```c
typedef int Tk_ErrorProc(
    ClientData clientData,
    XErrorEvent *errEventPtr);
```

The `clientData` parameter to `proc` is a copy of the `clientData` argument given to `Tcl_CreateErrorHandler` when the callback was created.
Typically, `clientData` points to a data structure containing application-specific information that is needed to deal with the error. `ErrEventPtr` is a pointer to the X error event. The procedure `proc` should return an integer value. If it returns 0 it means that `proc` handled the error completely and there is no need to take any other action for the error. If it returns non-zero it means `proc` was unable to handle the error.

If a value of NULL is specified for `proc`, all matching errors will be ignored: this will produce the same result as if a procedure had been specified that always returns 0.

If more than one handler matches a particular error, then they are invoked in turn. The handlers will be invoked in reverse order of creation: most recently declared handler first. If any handler returns 0, then subsequent (older) handlers will not be invoked. If no handler returns 0, then Tk invokes X’s default error handler, which prints an error message and aborts the program. If you wish to have a default handler that deals with errors that no other handler can deal with, then declare it first.

The X documentation states that “the error handler should not call any functions (directly or indirectly) on the display that will generate protocol requests or that will look for input events.” This restriction applies to handlers declared by `Tk_CreateErrorHandler`; disobey it at your own risk.

`Tk_DeleteErrorHandler` may be called to delete a previously-created error handler. The `handler` argument identifies the error handler, and should be a value returned by a previous call to `Tk_CreateErrorHandler`.

A particular error handler applies to errors resulting from protocol requests generated between the call to `Tk_CreateErrorHandler` and the call to `Tk_DeleteErrorHandler`. However, the actual callback to `proc` may not occur until after the `Tk_DeleteErrorHandler` call, due to buffering in the client and server. If an error event pertains to a protocol request made just before calling `Tk_DeleteErrorHandler`, then the error event may not have been processed before the
Tk_DeleteErrorHandler call. When this situation arises, Tk will save information about the handler and invoke the handler's proc later when the error event finally arrives. If an application wishes to delete an error handler and know for certain that all relevant errors have been processed, it should first call Tk_DeleteErrorHandler and then call XSync; this will flush out any buffered requests and errors, but will result in a performance penalty because it requires communication to and from the X server. After the XSync call Tk is guaranteed not to call any error handlers deleted before the XSync call.

For the Tk error handling mechanism to work properly, it is essential that application code never calls XSetErrorHandler directly; applications should use only Tk_CreateErrorHandler.

**KEYWORDS**
callback, error, event, handler

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME
Tk_Name, Tk_PathName, Tk_NameToWindow - convert between names and window tokens

SYNOPSIS
#include <tk.h>
Tk_Uid
Tk_Name(tkwin)
char *
Tk_PathName(tkwin)
Tk_Window
Tk_NameToWindow(interp, pathName, tkwin)

ARGUMENTS
DESCRIPTION
KEYWORDS
<table>
<thead>
<tr>
<th>Tk_Window tkwin (in)</th>
<th>Token for window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_Interp *interp (out)</td>
<td>Interpreter to use for error reporting.</td>
</tr>
<tr>
<td>const char *pathName (in)</td>
<td>Character string containing path name of window.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

Each window managed by Tk has two names, a short name that identifies a window among children of the same parent, and a path name that identifies the window uniquely among all the windows belonging to the same main window. The path name is used more often in Tk than the short name; many commands, like `bind`, expect path names as arguments.

The **Tk_Name** macro returns a window’s short name, which is the same as the `name` argument passed to **Tk_CreateWindow** when the window was created. The value is returned as a **Tk_Uid**, which may be used just like a string pointer but also has the properties of a unique identifier (see the manual entry for **Tk_GetUid** for details).

The **Tk_PathName** macro returns a hierarchical name for `tkwin`. Path names have a structure similar to file names in Unix but with dots between elements instead of slashes: the main window for an application has the path name "."; its children have names like ".a" and ".b"; their children have names like ".a.aa" and ".b.bb"; and so on. A window is considered to be a child of another window for naming purposes if the second window was named as the first window's `parent` when the first window was created. This is not always the same as the X window hierarchy. For example, a pop-up is created as a child of the root window, but its logical parent will usually be a window within the application.

The procedure **Tk_NameToWindow** returns the token for a window given its path name (the `pathName` argument) and another window
belonging to the same main window \((tkwin)\). It normally returns a token for the named window, but if no such window exists \texttt{Tk\_NameToWindow} leaves an error message in \texttt{interp->result} and returns NULL. The \textit{tkwin} argument to \texttt{Tk\_NameToWindow} is needed because path names are only unique within a single application hierarchy. If, for example, a single process has opened two main windows, each will have a separate naming hierarchy and the same path name might appear in each of the hierarchies. Normally \textit{tkwin} is the main window of the desired hierarchy, but this need not be the case: any window in the desired hierarchy may be used.

\textbf{KEYWORDS}

\textit{name, path name, token, window}
**NAME**

Tk_CreateEventHandler, Tk_DeleteEventHandler - associate procedure callback with an X event

**SYNOPSIS**

```c
#include <tk.h>
Tk_CreateEventHandler(tkwin, mask, proc, clientData)
Tk_DeleteEventHandler(tkwin, mask, proc, clientData)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tk_Window tkwin</strong> (in)</td>
<td>Token for window in which events may occur.</td>
</tr>
<tr>
<td><strong>unsigned long mask</strong> (in)</td>
<td>Bit-mask of events (such as ButtonPressMask) for which proc should be called.</td>
</tr>
<tr>
<td>*<em>Tk_EventProc <em>proc</em></em> (in)</td>
<td>Procedure to invoke whenever an event in mask occurs in the window given by tkwin.</td>
</tr>
<tr>
<td><strong>ClientData clientData</strong> (in)</td>
<td>Arbitrary one-word value to pass to proc.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**
**Tk_CreateEventHandler** arranges for *proc* to be invoked in the future whenever one of the event types specified by *mask* occurs in the window specified by *tkwin*. The callback to *proc* will be made by **Tk_HandleEvent**; this mechanism only works in programs that dispatch events through **Tk_HandleEvent** (or through other Tk procedures that call **Tk_HandleEvent**, such as **Tk_DoOneEvent** or **Tk_MainLoop**).

*Proc* should have arguments and result that match the type **Tk_EventProc**:

```c
typedef void Tk_EventProc(
    ClientData clientData,
    XEvent *eventPtr);
```

The *clientData* parameter to *proc* is a copy of the *clientData* argument given to **Tk_CreateEventHandler** when the callback was created. Typically, *clientData* points to a data structure containing application-specific information about the window in which the event occurred. *EventPtr* is a pointer to the X event, which will be one of the ones specified in the *mask* argument to **Tk_CreateEventHandler**.

**Tk_DeleteEventHandler** may be called to delete a previously-created event handler: it deletes the first handler it finds that is associated with *tkwin* and matches the *mask*, *proc*, and *clientData* arguments. If no such handler exists, then **Tk_HandleEvent** returns without doing anything. Although Tk supports it, it's probably a bad idea to have more than one callback with the same *mask*, *proc*, and *clientData* arguments. When a window is deleted all of its handlers will be deleted automatically; in this case there is no need to call **Tk_DeleteEventHandler**.

If multiple handlers are declared for the same type of X event on the same window, then the handlers will be invoked in the order they were created.
bind, callback, event, handler

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_CreateGenericHandler, Tk_DeleteGenericHandler - associate procedure callback with all X events

SYNOPSIS

#include <tk.h>
Tk_CreateGenericHandler(proc, clientData)
Tk_DeleteGenericHandler(proc, clientData)

ARGUMENTS

Tk_GenericProc *proc (in)  Procedure to invoke whenever any X event occurs on any display.
ClientData clientData (in)  Arbitrary one-word value to pass to proc.

DESCRIPTION

Tk_CreateGenericHandler arranges for proc to be invoked in the future whenever any X event occurs. This mechanism is not intended for dispatching X events on windows managed by Tk (you should use Tk_CreateEventHandler for this purpose). Tk_CreateGenericHandler is intended for other purposes, such as tracing X events, monitoring events on windows not owned by Tk, accessing X-related libraries that were not originally designed for use with Tk, and so on.

The callback to proc will be made by Tk_HandleEvent; this mechanism
only works in programs that dispatch events through Tk_HandleEvent (or through other Tk procedures that call Tk_HandleEvent, such as Tk_DoOneEvent or Tk_MainLoop).

Proc should have arguments and result that match the type Tk_GenericProc:

```c
typedef int Tk_GenericProc(
    ClientData clientData,
    XEvent *eventPtr);
```

The clientData parameter to proc is a copy of the clientData argument given to Tk_CreateGenericHandler when the callback was created. Typically, clientData points to a data structure containing application-specific information about how to handle events. EventPtr is a pointer to the X event.

Whenever an X event is processed by Tk_HandleEvent, proc is called. The return value from proc is normally 0. A non-zero return value indicates that the event is not to be handled further; that is, proc has done all processing that is to be allowed for the event.

If there are multiple generic event handlers, each one is called for each event, in the order in which they were established.

Tk_DeleteGenericHandler may be called to delete a previously-created generic event handler: it deletes each handler it finds that matches the proc and clientData arguments. If no such handler exists, then Tk_DeleteGenericHandler returns without doing anything. Although Tk supports it, it's probably a bad idea to have more than one callback with the same proc and clientData arguments.

Establishing a generic event handler does nothing to ensure that the process will actually receive the X events that the handler wants to process. For example, it is the caller's responsibility to invoke XSelectInput to select the desired events, if that is necessary.
KEYWORDS

bind, callback, event, handler

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1992-1994 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
**NAME**

Tk_GetDash - convert from string to valid dash structure.

**SYNOPSIS**

```c
#include <tk.h>
int Tk_GetDash(interp, string, dashPtr)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tcl_Interp *interp (in)</td>
<td>Interpreter to use for error reporting.</td>
</tr>
<tr>
<td>const char * string (in)</td>
<td>Textual value to be converted.</td>
</tr>
<tr>
<td>Tk_Dash *dashPtr (out)</td>
<td>Points to place to store the dash pattern value converted from string.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

These procedure parses the string and fills in the result in the Tk_Dash structure. The string can be a list of integers or a character string containing only “.,-,” or spaces. If all goes well, TCL_OK is returned. If string does not have the proper syntax then TCL_ERROR is returned, an error message is left in the interpreter's result, and nothing is stored at *dashPtr.
The first possible syntax is a list of integers. Each element represents the number of pixels of a line segment. Only the odd segments are drawn using the “outline” color. The other segments are drawn transparent.

The second possible syntax is a character list containing only 5 possible characters “.,_-”. The space can be used to enlarge the space between other line elements, and can not occur as the first position in the string. Some examples:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-dash</td>
<td>.</td>
<td>=</td>
<td>-dash</td>
</tr>
<tr>
<td>-dash</td>
<td>-</td>
<td>=</td>
<td>-dash</td>
</tr>
<tr>
<td>-dash</td>
<td>-</td>
<td>=</td>
<td>-dash</td>
</tr>
<tr>
<td>-dash</td>
<td>-</td>
<td>=</td>
<td>-dash</td>
</tr>
<tr>
<td>-dash</td>
<td>{ . }</td>
<td>=</td>
<td>-dash</td>
</tr>
<tr>
<td>-dash</td>
<td>,</td>
<td>=</td>
<td>-dash</td>
</tr>
</tbody>
</table>

The main difference of this syntax with the previous is that it is shape-conserving. This means that all values in the dash list will be multiplied by the line width before display. This assures that “.” will always be displayed as a dot and “-” always as a dash regardless of the line width.

On systems where only a limited set of dash patterns, the dash pattern will be displayed as the most close dash pattern that is available. For example, on Windows only the first 4 of the above examples are available. The last 2 examples will be displayed identically as the first one.

**KEYWORDS**

dash, conversion
NAME
Tk_CreateItemType, Tk_GetItemTypes - define new kind of canvas item

SYNOPSIS
#include <tk.h>
Tk_CreateItemType(typePtr)
Tk_ItemType *
Tk_GetItemTypes()

ARGUMENTS

INTRODUCTION

DATA STRUCTURES

NAME

ITEMSIZE
CREATEPROC
CONFIGSPECS
CONFIGPROC
COORDPROC
DELETEPROC
DISPLAYPROC AND ALWAYSREDRAW
POINTPROC
AREAPROC
POSTSCRIPTPROC
SCALEPROC
TRANSLATEPROC
INDEXPROC
ICURSORPROC
SELECTIONPROC
INSERTPROC
DCHARSPROC
SEE ALSO
KEYWORDS
NAME

Tk_CreateItemType, Tk_GetItemTypes - define new kind of canvas item

SYNOPSIS

#include <tk.h>
Tk_CreateItemType(typePtr)
Tk_ItemType *
Tk_GetItemTypes()

ARGUMENTS

Tk_ItemType *typePtr (in) Structure that defines the new type of canvas item.

INTRODUCTION

Tk_CreateItemType is invoked to define a new kind of canvas item described by the typePtr argument. An item type corresponds to a particular value of the type argument to the create widget command for canvases, and the code that implements a canvas item type is called a type manager. Tk defines several built-in item types, such as rectangle and text and image, but Tk_CreateItemType allows additional item types to be defined. Once Tk_CreateItemType returns, the new item type may be used in new or existing canvas widgets just like the built-in item types.

Tk_GetItemTypes returns a pointer to the first in the list of all item types currently defined for canvases. The entries in the list are linked together through their nextPtr fields, with the end of the list marked by a NULL nextPtr.

You may find it easier to understand the rest of this manual entry by looking at the code for an existing canvas item type such as bitmap (file tkCanvBmap.c) or text (tkCanvText.c). The easiest way to create a new type manager is to copy the code for an existing type and modify it for
the new type.

Tk provides a number of utility procedures for the use of canvas type managers, such as `Tk_CanvasCoords` and `Tk_CanvasPsColor`; these are described in separate manual entries.

## DATA STRUCTURES

A type manager consists of a collection of procedures that provide a standard set of operations on items of that type. The type manager deals with three kinds of data structures. The first data structure is a `Tk_ItemType`; it contains information such as the name of the type and pointers to the standard procedures implemented by the type manager:

```c
typedef struct Tk_ItemType {
    char *name;
    int itemSize;
    Tk_ItemCreateProc *createProc;
    Tk_ConfigSpec *configSpecs;
    Tk_ItemConfigureProc *configProc;
    Tk_ItemCoordProc *coordProc;
    Tk_ItemDeleteProc *deleteProc;
    Tk_ItemDisplayProc *displayProc;
    int alwaysRedraw;
    Tk_ItemPointProc *pointProc;
    Tk_ItemAreaProc *areaProc;
    Tk_ItemPostscriptProc *postscriptProc;
    Tk_ItemScaleProc *scaleProc;
    Tk_ItemTranslateProc *translateProc;
    Tk_ItemIndexProc *indexProc;
    Tk_ItemCursorProc *icursorProc;
    Tk_ItemSelectionProc *selectionProc;
    Tk_ItemInsertProc *insertProc;
    Tk_ItemDCharsProc *dCharsProc;
    Tk_ItemType *nextPtr;
} Tk_ItemType;
```
The fields of a Tk_ItemType structure are described in more detail later in this manual entry. When Tk_CreateItemType is called, its typePtr argument must point to a structure with all of the fields initialized except nextPtr, which Tk sets to link all the types together into a list. The structure must be in permanent memory (either statically allocated or dynamically allocated but never freed); Tk retains a pointer to this structure.

The second data structure manipulated by a type manager is an item record. For each item in a canvas there exists one item record. All of the items of a given type generally have item records with the same structure, but different types usually have different formats for their item records. The first part of each item record is a header with a standard structure defined by Tk via the type Tk_Item; the rest of the item record is defined by the type manager. A type manager must define its item records with a Tk_Item as the first field. For example, the item record for bitmap items is defined as follows:

```c
typedef struct BitmapItem {
    Tk_Item header;
    double x, y;
    Tk_Anchor anchor;
    Pixmap bitmap;
    XColor *fgColor;
    XColor *bgColor;
    GC gc;
} BitmapItem;
```

The header substructure contains information used by Tk to manage the item, such as its identifier, its tags, its type, and its bounding box. The fields starting with x belong to the type manager: Tk will never read or write them. The type manager should not need to read or write any of the fields in the header except for four fields whose names are x1, y1, x2, and y2. These fields give a bounding box for the items using integer canvas coordinates: the item should not cover any pixels with x-coordinate lower than x1 or y-coordinate lower than y1, nor should it
cover any pixels with x-coordinate greater than or equal to x2 or y-coordinate greater than or equal to y2. It is up to the type manager to keep the bounding box up to date as the item is moved and reconfigured.

Whenever Tk calls a procedure in a type manager it passes in a pointer to an item record. The argument is always passed as a pointer to a Tk_Item; the type manager will typically cast this into a pointer to its own specific type, such as BitmapItem.

The third data structure used by type managers has type Tk_Canvas; it serves as an opaque handle for the canvas widget as a whole. Type managers need not know anything about the contents of this structure. A Tk_Canvas handle is typically passed in to the procedures of a type manager, and the type manager can pass the handle back to library procedures such as Tk_CanvasTkwin to fetch information about the canvas.

**NAME**

This section and the ones that follow describe each of the fields in a Tk_ItemType structure in detail. The name field provides a string name for the item type. Once Tk_CreateImageType returns, this name may be used in create widget commands to create items of the new type. If there already existed an item type by this name then the new item type replaces the old one.

**ITEMSIZE**

`typePtr->itemSize` gives the size in bytes of item records of this type, including the Tk_Item header. Tk uses this size to allocate memory space for items of the type. All of the item records for a given type must have the same size. If variable length fields are needed for an item (such as a list of points for a polygon), the type manager can allocate a separate object of variable length and keep a pointer to it in the item record.
**CREATEPROC**

`typePtr->createProc` points to a procedure for Tk to call whenever a new item of this type is created. `typePtr->createProc` must match the following prototype:

```c
typedef int Tk_ItemCreateProc(
    Tcl_Interp  *interp,
    Tk_Canvas  canvas,
    Tk_Item  *itemPtr,
    int  objc,
    Tcl_Obj*  const  objv[]);
```

The `interp` argument is the interpreter in which the canvas's `create` widget command was invoked, and `canvas` is a handle for the canvas widget. `itemPtr` is a pointer to a newly-allocated item of size `typePtr->itemSize`. Tk has already initialized the item's header (the first `sizeof(Tk_ItemType)` bytes). The `objc` and `objv` arguments describe all of the arguments to the `create` command after the `type` argument. For example, in the widget command

```
c  create rectangle 10 20 50 50 -fill black
```

`objc` will be 6 and `objv[0]` will contain the integer object 10.

`createProc` should use `objc` and `objv` to initialize the type-specific parts of the item record and set an initial value for the bounding box in the item's header. It should return a standard Tcl completion code and leave an error message in `interp->result` if an error occurs. If an error occurs Tk will free the item record, so `createProc` must be sure to leave the item record in a clean state if it returns an error (e.g., it must free any additional memory that it allocated for the item).
Each type manager must provide a standard table describing its configuration options, in a form suitable for use with \texttt{Tk ConfigureWidget}. This table will normally be used by \texttt{typePtr->createProc} and \texttt{typePtr->configProc}, but Tk also uses it directly to retrieve option information in the \texttt{itemcget} and \texttt{itemconfigure} widget commands. \texttt{typePtr->configSpecs} must point to the configuration table for this type. Note: Tk provides a custom option type \texttt{tk CanvasTagsOption} for implementing the \texttt{-tags} option; see an existing type manager for an example of how to use it in \texttt{configSpecs}.

\begin{center}
\textbf{CONFIGPROC}
\end{center}

\texttt{typePtr->configProc} is called by Tk whenever the \texttt{itemconfigure} widget command is invoked to change the configuration options for a canvas item. This procedure must match the following prototype:

```
typedef int Tk_ItemConfigureProc(
    \texttt{Tcl Interp} * interp,
    \texttt{Tk Canvas} canvas,
    \texttt{Tk Item} * itemPtr,
    int objc,
    \texttt{Tcl Obj*} const objv[],
    int flags);
```

The \texttt{interp} objument identifies the interpreter in which the widget command was invoked, \texttt{canvas} is a handle for the canvas widget, and \texttt{itemPtr} is a pointer to the item being configured. \texttt{objc} and \texttt{objv} contain the configuration options. For example, if the following command is invoked:

```
.c itemconfigure 2 -fill red -outline black
```

\texttt{objc} is 4 and \texttt{objv} contains the string objects \texttt{-fill} through \texttt{black}. \texttt{objc} will always be an even value. The \texttt{flags} argument contains flags to pass
to **Tk_ConfigureWidget**; currently this value is always **TK_CONFIG_ARGV_ONLY** when Tk invokes `typePtr->configProc`, but the type manager's `createProc` procedure will usually invoke `configProc` with different flag values.

`typePtr->configProc` returns a standard Tcl completion code and leaves an error message in `interp->result` if an error occurs. It must update the item's bounding box to reflect the new configuration options.

---

### COORDPROC

`typePtr->coordProc` is invoked by Tk to implement the **coords** widget command for an item. It must match the following prototype:

```c
typedef int Tk_ItemCoordProc(
    Tcl_Interp *interp,
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int objc,
    Tcl_Obj* const objv[]);
```

The arguments `interp`, `canvas`, and `itemPtr` all have the standard meanings, and `objc` and `objv` describe the coordinate arguments. For example, if the following widget command is invoked:

```c
.c coords 2 30 90
```

`objc` will be 2 and `objv` will contain the integer objects 30 and 90.

The `coordProc` procedure should process the new coordinates, update the item appropriately (e.g., it must reset the bounding box in the item's header), and return a standard Tcl completion code. If an error occurs, `coordProc` must leave an error message in `interp->result`.

---

### DELETEPROC
typePtr->deleteProc is invoked by Tk to delete an item and free any resources allocated to it. It must match the following prototype:

```c
typedef void Tk_ItemDeleteProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    Display *display);
```

The `canvas` and `itemPtr` arguments have the usual interpretations, and `display` identifies the X display containing the canvas. `deleteProc` must free up any resources allocated for the item, so that Tk can free the item record. `deleteProc` should not actually free the item record; this will be done by Tk when `deleteProc` returns.

**DISPLAYPROC AND ALWAYSREDRAW**

typePtr->displayProc is invoked by Tk to redraw an item on the screen. It must match the following prototype:

```c
typedef void Tk_ItemDisplayProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    Display *display,
    Drawable dst,
    int x,
    int y,
    int width,
    int height);
```

The `canvas` and `itemPtr` arguments have the usual meaning. `display` identifies the display containing the canvas, and `dst` specifies a drawable in which the item should be rendered; typically this is an off-screen pixmap, which Tk will copy into the canvas's window once all relevant items have been drawn. `x`, `y`, `width`, and `height` specify a
rectangular region in canvas coordinates, which is the area to be redrawn; only information that overlaps this area needs to be redrawn. Tk will not call *displayProc* unless the item's bounding box overlaps the redraw area, but the type manager may wish to use the redraw area to optimize the redisplay of the item.

Because of scrolling and the use of off-screen pixmaps for double-buffered redisplay, the item's coordinates in *dst* will not necessarily be the same as those in the canvas. *displayProc* should call *Tk_CanvasDrawableCoords* to transform coordinates from those of the canvas to those of *dst*.

Normally an item's *displayProc* is only invoked if the item overlaps the area being displayed. However, if *typePtr-* > *alwaysRedraw* has a non-zero value, then *displayProc* is invoked during every redisplay operation, even if the item does not overlap the area of redisplay. *alwaysRedraw* should normally be set to 0; it is only set to 1 in special cases such as window items that need to be unmapped when they are off-screen.

**POINTPROC**

*typePtr-* > *pointProc* is invoked by Tk to find out how close a given point is to a canvas item. Tk uses this procedure for purposes such as locating the item under the mouse or finding the closest item to a given point. The procedure must match the following prototype:

```c
typedef double Tk_ItemPointProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    double *pointPtr);
```

canvas and itemPtr have the usual meaning. pointPtr points to an array of two numbers giving the x and y coordinates of a point. pointProc must return a real value giving the distance from the point to the item, or 0 if the point lies inside the item.
typePtr->areaProc is invoked by Tk to find out the relationship between an item and a rectangular area. It must match the following prototype:

```c
typedef int Tk_ItemAreaProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    double *rectPtr);
```

`canvas` and `itemPtr` have the usual meaning. `rectPtr` points to an array of four real numbers; the first two give the x and y coordinates of the upper left corner of a rectangle, and the second two give the x and y coordinates of the lower right corner. `areaProc` must return -1 if the item lies entirely outside the given area, 0 if it lies partially inside and partially outside the area, and 1 if it lies entirely inside the area.

**POSTSCRIPTPROC**

typePtr->postscriptProc is invoked by Tk to generate Postscript for an item during the `postscript` widget command. If the type manager is not capable of generating Postscript then `typePtr->postscriptProc` should be NULL. The procedure must match the following prototype:

```c
typedef int Tk_ItemPostscriptProc(
    Tcl_Interp *interp,
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int prepass);
```

The `interp`, `canvas`, and `itemPtr` arguments all have standard meanings; `prepass` will be described below. If `postscriptProc` completes successfully, it should append Postscript for the item to the information in `interp->result` (e.g. by calling `Tcl_AppendResult`, not `Tcl_SetResult`)
and return `TCL_OK`. If an error occurs, `postscriptProc` should clear the result and replace its contents with an error message; then it should return `TCL_ERROR`.

Tk provides a collection of utility procedures to simplify `postscriptProc`. For example, `Tk_CanvasPsColor` will generate Postscript to set the current color to a given Tk color and `Tk_CanvasPsFont` will set up font information. When generating Postscript, the type manager is free to change the graphics state of the Postscript interpreter, since Tk places `gsave` and `grestore` commands around the Postscript for the item. The type manager can use canvas x coordinates directly in its Postscript, but it must call `Tk_CanvasPsY` to convert y coordinates from the space of the canvas (where the origin is at the upper left) to the space of Postscript (where the origin is at the lower left).

In order to generate Postscript that complies with the Adobe Document Structuring Conventions, Tk actually generates Postscript in two passes. It calls each item's `postscriptProc` in each pass. The only purpose of the first pass is to collect font information (which is done by `Tk_CanvasPsFont`); the actual Postscript is discarded. Tk sets the `prepass` argument to `postscriptProc` to 1 during the first pass; the type manager can use `prepass` to skip all Postscript generation except for calls to `Tk_CanvasPsFont`. During the second pass `prepass` will be 0, so the type manager must generate complete Postscript.

### SCALEPROC

`typePtr->scaleProc` is invoked by Tk to rescale a canvas item during the `scale` widget command. The procedure must match the following prototype:

```c
typedef void Tk_ItemScaleProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    double originX,
    double originY,
    double scaleX,
    double scaleY,
);
```
The `canvas` and `itemPtr` arguments have the usual meaning. `originX` and `originY` specify an origin relative to which the item is to be scaled, and `scaleX` and `scaleY` give the x and y scale factors. The item should adjust its coordinates so that a point in the item that used to have coordinates \( x \) and \( y \) will have new coordinates \( x' \) and \( y' \), where

\[
x' = \text{originX} + \text{scaleX} \times (x - \text{originX})
\]

\[
y' = \text{originY} + \text{scaleY} \times (y - \text{originY})
\]

`scaleProc` must also update the bounding box in the item's header.

**TRANSLATEPROC**

`typePtr->translateProc` is invoked by Tk to translate a canvas item during the `move` widget command. The procedure must match the following prototype:

```c
typedef void Tk_ItemTranslateProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    double deltaX,
    double deltaY);
```

The `canvas` and `itemPtr` arguments have the usual meaning, and `deltaX` and `deltaY` give the amounts that should be added to each \( x \) and \( y \) coordinate within the item. The type manager should adjust the item's coordinates and update the bounding box in the item's header.

**INDEXPROC**

`typePtr->indexProc` is invoked by Tk to translate a string index
specification into a numerical index, for example during the index widget command. It is only relevant for item types that support indexable text; `typePtr->indexProc` may be specified as NULL for non-textual item types. The procedure must match the following prototype:

```c
typedef int Tk_ItemIndexProc(
    Tcl_Interp *interp,
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    char indexString,
    int *indexPtr);
```

The `interp`, `canvas`, and `itemPtr` arguments all have the usual meaning. `indexString` contains a textual description of an index, and `indexPtr` points to an integer value that should be filled in with a numerical index. It is up to the type manager to decide what forms of index are supported (e.g., numbers, `insert, sel.first, end`, etc.). `indexProc` should return a Tcl completion code and set `interp->result` in the event of an error.

**ICURSORPROC**

typePtr->icursorProc is invoked by Tk during the icursor widget command to set the position of the insertion cursor in a textual item. It is only relevant for item types that support an insertion cursor; `typePtr->icursorProc` may be specified as NULL for item types that do not support an insertion cursor. The procedure must match the following prototype:

```c
typedef void Tk_ItemCursorProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int index);
```

`canvas` and `itemPtr` have the usual meanings, and `index` is an index into
the item's text, as returned by a previous call to typePtr->insertProc. The type manager should position the insertion cursor in the item just before the character given by index. Whether or not to actually display the insertion cursor is determined by other information provided by Tk_CanvasGetTextInfo.

**SELECTIONPROC**

typePtr->selectionProc is invoked by Tk during selection retrievals; it must return part or all of the selected text in the item (if any). It is only relevant for item types that support text; typePtr->selectionProc may be specified as NULL for non-textual item types. The procedure must match the following prototype:

```c
typedef int Tk_ItemSelectionProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int offset,
    char *buffer,
    int maxBytes);
```

canvas and itemPtr have the usual meanings. offset is an offset in bytes into the selection where 0 refers to the first byte of the selection; it identifies the first character that is to be returned in this call. buffer points to an area of memory in which to store the requested bytes, and maxBytes specifies the maximum number of bytes to return. selectionProc should extract up to maxBytes characters from the selection and copy them to maxBytes; it should return a count of the number of bytes actually copied, which may be less than maxBytes if there are not offset+maxBytes bytes in the selection.

**INSERTPROC**

typePtr->insertProc is invoked by Tk during the insert widget command to insert new text into a canvas item. It is only relevant for item types that support text; typePtr->insertProc may be specified as NULL for
non-textual item types. The procedure must match the following prototype:

```c
typedef void Tk_ItemInsertProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int index,
    char *string);
```

canvas and itemPtr have the usual meanings. index is an index into the item's text, as returned by a previous call to typePtr->insertProc, and string contains new text to insert just before the character given by index. The type manager should insert the text and recompute the bounding box in the item's header.

**DCHARSPROC**

typePtr->dCharsProc is invoked by Tk during the dchars widget command to delete a range of text from a canvas item. It is only relevant for item types that support text; typePtr->dCharsProc may be specified as NULL for non-textual item types. The procedure must match the following prototype:

```c
typedef void Tk_ItemDCharsProc(
    Tk_Canvas canvas,
    Tk_Item *itemPtr,
    int first,
    int last);
```

canvas and itemPtr have the usual meanings. first and last give the indices of the first and last bytes to be deleted, as returned by previous calls to typePtr->indexProc. The type manager should delete the specified characters and update the bounding box in the item's header.
SEE ALSO

Tk_CanvasPsY, Tk_CanvasTextInfo, Tk_CanvasTkwin

KEYWORDS

canvas, focus, item type, selection, type manager

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1994-1995 Sun Microsystems, Inc.
NAME
Ttk_MakeBox, Ttk_PadBox, Ttk_ExpandBox, Ttk_PackBox,
Ttk_StickBox, Ttk_PlaceBox, Ttk_BoxContains,
Ttk_MakePadding, Ttk_UniformPadding, Ttk_AddPadding,
Ttk_RelievePadding, Ttk_GetPaddingFromObj,
Ttk_GetBorderFromObj, Ttk_GetStickyFromObj - Tk themed
geometry utilities

SYNOPSIS
#include <tkTheme.h>

Ttk_Box
Ttk_MakeBox(int x, int y, int width, int height);
Ttk_Box
Ttk_PadBox(Ttk_Box parcel, Ttk_Padding padding);
Ttk_Box
Ttk_ExpandBox(Ttk_Box parcel, Ttk_Padding padding);
Ttk_Box
Ttk_PackBox(Ttk_Box *cavity, int width, int height, Ttk_Side side);
Ttk_Box
Ttk_StickBox(Ttk_Box parcel, int width, int height, unsigned sticky);
Ttk_Box
Ttk_PlaceBox(Ttk_Box *cavity, int width, int height, Ttk_Side side, unsigned sticky);
Ttk_Box
Ttk_AnchorBox(Ttk_Box parcel, int width, int height,
Ttk_Anchor anchor);
Ttk_Padding
Ttk_MakePadding(short left, short top, short right, short bottom);
Ttk_Padding
Ttk_UniformPadding (short border);
Ttk_Padding
Ttk_AddPadding (Ttk_Padding padding1, Ttk_Padding padding2);
Ttk_Padding
Ttk_RelievePadding (Ttk_Padding padding, int relief);
int
Ttk_BoxContains (Ttk_Box box, int x, int y);
int
Ttk_GetPaddingFromObj (Tcl_Interp *interp, Tk_Window tkwin, Tcl_Obj *objPtr, Ttk_Padding *padding_rtn);
int
Ttk_GetBorderFromObj (Tcl_Interp *interp, Tcl_Obj *objPtr, Ttk_Padding *padding_rtn);
int
Ttk_GetStickyFromObj (Tcl_Interp *interp, Tcl_Obj *objPtr, int *sticky_rtn);

ARGUMENTS
BOXES
PADDLING
CONVERSION ROUTINES
SEE ALSO
KEYWORDS

NAME
Ttk_MakeBox, Ttk_PadBox, Ttk_ExpandBox, Ttk_PackBox,
Ttk_ResizeBox, Ttk_PlaceBox, Ttk_BoxContains, Ttk_MakePadding,
Ttk_UniformPadding, Ttk_AddPadding, Ttk_RelievePadding,
Ttk_GetPaddingFromObj, Ttk_GetBorderFromObj,
Ttk_GetStickyFromObj - Tk themed geometry utilities

SYNOPSIS

#include <tkTheme.h>

Ttk_Box
Ttk_MakeBox (int x, int y, int width, int height);
Ttk_Box
Ttk_PadBox(Ttk_Box parcel, Ttk_Padding padding);

Ttk_Box
Ttk_ExpandBox(Ttk_Box parcel, Ttk_Padding padding);

Ttk_Box
Ttk_PackBox(Ttk_Box *cavity, int width, int height, Ttk_Side side);

Ttk_Box
Ttk_StickBox(Ttk_Box parcel, int width, int height, unsigned sticky);

Ttk_Box
Ttk_PlaceBox(Ttk_Box *cavity, int width, int height, Ttk_Side side, unsigned sticky);

Ttk_Box
Ttk_AnchorBox(Ttk_Box parcel, int width, int height, Tk_Anchor anchor);

Ttk_Padding
Ttk_MakePadding(short left, short top, short right, short bottom);

Ttk_Padding
Ttk_UniformPadding(short border);

Ttk_Padding
Ttk_AddPadding(Ttk_Padding padding1, Ttk_Padding padding2);

Ttk_Padding
Ttk_RelievePadding(Ttk_Padding padding, int relief);

int
Ttk_BoxContains(Ttk_Box box, int x, int y);

int
Ttk_GetPaddingFromObj(Tcl_Interp *interp, Tk_Window tkwin, Tcl_Obj *objPtr, Ttk_Padding *padding_rtn);

int Ttk_GetBorderFromObj(Tcl_Interp *interp, Tcl_Obj *objPtr, Ttk_Padding *padding_rtn);

int Ttk_GetStickyFromObj(Tcl_Interp *interp, Tcl_Obj *objPtr, int *sticky_rtn);

### ARGUMENTS

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Anchor anchor (in)</td>
<td>One of the symbolic constants TK_ANCHOR_N, TK_ANCHOR_NE, etc. See Tk_GetAnchorFromObj(3).</td>
</tr>
<tr>
<td>Ttk_Box * cavity (in/out)</td>
<td>A rectangular region from which a parcel is allocated.</td>
</tr>
<tr>
<td>short border (in)</td>
<td>Extra padding (in pixels) to add uniformly to each side of a region.</td>
</tr>
<tr>
<td>short bottom (in)</td>
<td>Extra padding (in pixels) to add to the bottom of a region.</td>
</tr>
<tr>
<td>Ttk_Box box (in)</td>
<td></td>
</tr>
<tr>
<td>Ttk_Box * box_rtn (out)</td>
<td>Specifies a rectangular region.</td>
</tr>
<tr>
<td>int height (in)</td>
<td>The height in pixels of a</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Tcl_Interp * interp</strong> (in)</td>
<td>Used to store error messages.</td>
</tr>
<tr>
<td>int <strong>left</strong> (in)</td>
<td>Extra padding (in pixels) to add to the left side of a region.</td>
</tr>
<tr>
<td><strong>Tcl_Obj * objPtr</strong> (in)</td>
<td>String value contains a symbolic name to be converted to an enumerated value or bitmask. Internal rep may be modified to cache corresponding value.</td>
</tr>
<tr>
<td><strong>Ttk_Padding padding</strong> (in)</td>
<td>Extra padding to add on the inside of a region.</td>
</tr>
<tr>
<td><strong>Ttk_Padding * padding_rtn</strong> (out)</td>
<td>Extra padding to add on the inside of a region.</td>
</tr>
<tr>
<td><strong>Ttk_Box parcel</strong> (in)</td>
<td>A rectangular region, allocated from a cavity.</td>
</tr>
<tr>
<td>int <strong>relief</strong> (in)</td>
<td>One of the standard Tk relief options (TK_RELIEF_RAISED, TK_RELIEF_SUNKEN, etc.). See <a href="#">Tk_GetReliefFromObj</a>.</td>
</tr>
<tr>
<td>short <strong>right</strong> (in)</td>
<td>Extra padding (in pixels) to add to the right side of a region.</td>
</tr>
<tr>
<td><strong>Ttk_Side side</strong> (in)</td>
<td>One of TTK_SIDE_LEFT, TTK_SIDE_TOP,</td>
</tr>
</tbody>
</table>
unsigned **sticky** (in)  
A bitmask containing one or more of the bits  
`TTK_STICK_W` (west, or left), `TTK_STICK_E` (east, or right), `TTK_STICK_N` (north, or top), and  
`TTK_STICK_S` (south, or bottom). `TTK_FILL_X` is defined as a synonym for  
`(TTK_STICK_W|TTK_STICK_E)`,  
`TTK_FILL_Y` is a synonym for  
`(TTK_STICK_N|TTK_STICK_S)`,  
and `TTK_FILL_BOTH` and `TTK_STICK_ALL` are synonyms for  
`(TTK_FILL_X|TTK_FILL_Y)`.  
See also: `grid(n)`.

Tk_Window **tkwin** (in)  
Window whose screen geometry determines the conversion between  
absolute units and pixels.

short **top** (in)  
Extra padding at the top of a region.

int **width** (in)  
The width in pixels of a region.

int **x** (in)  
X coordinate of upper-left corner of region.

int **y** (in)  
Y coordinate of upper-left...
BOXES

The **Ttk_Box** structure represents a rectangular region of a window:

```c
typedef struct {
    int x;
    int y;
    int width;
    int height;
} Ttk_Box;
```

All coordinates are relative to the window.

**Ttk_MakeBox** is a convenience routine that constructs a **Ttk_Box** structure representing a region `width` pixels wide, `height` pixels tall, at the specified `x`, `y` coordinates.

**Ttk_PadBox** returns a new box located inside the specified `parcel`, shrunken according to the left, top, right, and bottom margins specified by `padding`.

**Ttk_ExpandBox** is the inverse of **Ttk_PadBox**: it returns a new box surrounding the specified `parcel`, expanded according to the left, top, right, and bottom margins specified by `padding`.

**Ttk_PackBox** allocates a parcel `width` by `height` pixels wide on the specified `side` of the `cavity`, and shrinks the `cavity` accordingly.

**Ttk_StickBox** places a box with the requested `width` and `height` inside the `parcel` according to the `sticky` bits.

**Ttk_PlaceBox** combines **Ttk_PackBox** and **Ttk_StickBox**: it allocates a parcel on the specified `side` of the `cavity`, places a box of the requested size inside the parcel according to `sticky`, and shrinks the
cavity.

**Ttk_AnchorBox** places a box with the requested width and height inside the parcel according to the specified anchor option.

**Ttk_BoxContains** tests if the specified x, y coordinate lies within the rectangular region box.

---

**PADDDING**

The **Ttk_Padding** structure is used to represent borders, internal padding, and external margins:

```c
typedef struct {
    short left;
    short top;
    short right;
    short bottom;
} Ttk_Padding;
```

**Ttk_MakePadding** is a convenience routine that constructs a **Ttk_Padding** structure with the specified left, top, right, and bottom components.

**Ttk_UniformPadding** constructs a **Ttk_Padding** structure with all components equal to the specified border.

**Ttk_AddPadding** adds two **Ttk_Padding**s together and returns a combined padding containing the sum of the individual padding components.

**Ttk_RelievePadding** adds an extra 2 pixels of padding to padding according to the specified relief. If relief is **TK_RELIEF_SUNKEN**, adds two pixels at the top and left so the inner region is shifted down and to the left. If it is **TK_RELIEF_RAISED**, adds two pixels at the bottom and right so the inner region is shifted up and to the right. Otherwise, adds 1 pixel on all sides. This is typically used in element geometry procedures.
to simulate a “pressed-in” look for pushbuttons.

### CONVERSION ROUTINES

**Ttk_GetPaddingFromObj** converts the string in *objPtr* to a **Ttk_Padding** structure. The string representation is a list of up to four length specifications “left top right bottom”. If fewer than four elements are specified, *bottom* defaults to *top*, *right* defaults to *left*, and *top* defaults to *left*. See **Tk_GetPixelsFromObj(3)** for the syntax of length specifications.

**Ttk_GetBorderFromObj** is the same as **Ttk_GetPaddingFromObj** except that the lengths are specified as integers (i.e., resolution-dependant values like 3m are not allowed).

**Ttk_GetStickyFromObj** converts the string in *objPtr* to a *sticky* bitmask. The string contains zero or more of the characters *n*, *s*, *e*, or *w*.

### SEE ALSO

**Tk_GetReliefFromObj**, **Tk_GetPixelsFromObj**, **Tk_GetAnchorFromObj**

### KEYWORDS

geometry, padding, margins, box, region, sticky, relief
NAME
Tk_CreatePhotoImageFormat - define new file format for photo images

SYNOPSIS
#include <tk.h>
Tk_CreatePhotoImageFormat(formatPtr)

ARGUMENTS

DESCRIPTION
NAME
FILEMATCHPROC
STRINGMATCHPROC
FILEReadPROC
STRINGREADPROC
FILEWritePROC
STRINGWRITEPROC

LEGACY INTERFACE SUPPORT
SEE ALSO

KEYWORDS
**DESCRIPTION**

**Tk_CreatePhotoImageFormat** is invoked to define a new file format for image data for use with photo images. The code that implements an image file format is called an image file format handler, or handler for short. The photo image code maintains a list of handlers that can be used to read and write data to or from a file. Some handlers may also support reading image data from a string or converting image data to a string format. The user can specify which handler to use with the `-format` image configuration option or the `-format` option to the read and write photo image subcommands.

An image file format handler consists of a collection of procedures plus a Tk_PhotoImageFormat structure, which contains the name of the image file format and pointers to six procedures provided by the handler to deal with files and strings in this format. The Tk_PhotoImageFormat structure contains the following fields:

```c
typedef struct Tk_PhotoImageFormat {
    char *name;
    Tk_ImageFileMatchProc *fileMatchProc;
    Tk_ImageStringMatchProc *stringMatchProc;
    Tk_ImageFileReadProc  *fileReadProc;
    Tk_ImageStringReadProc *stringReadProc;
    Tk_ImageFileWriteProc *fileWriteProc;
    Tk_ImageStringWriteProc *stringWriteProc;
} Tk_PhotoImageFormat;
```

The handler need not provide implementations of all six procedures. For example, the procedures that handle string data would not be provided for a format in which the image data are stored in binary, and could therefore contain null characters. If any procedure is not implemented, the corresponding pointer in the Tk_PhotoImageFormat structure should be set to NULL. The handler must provide the `fileMatchProc` procedure if it provides the `fileReadProc` procedure, and the
stringMatchProc procedure if it provides the stringReadProc procedure.

**NAME**

formatPtr->name provides a name for the image type. Once Tk_CreatePhotoImageFormat returns, this name may be used in the -format photo image configuration and subcommand option. The manual page for the photo image (photo(n)) describes how image file formats are chosen based on their names and the value given to the -format option. The first character of formatPtr->name must not be an uppercase character from the ASCII character set (that is, one of the characters A-Z). Such names are used only for legacy interface support (see below).

**FILEMATCHPROC**

formatPtr->fileMatchProc provides the address of a procedure for Tk to call when it is searching for an image file format handler suitable for reading data in a given file. formatPtr->fileMatchProc must match the following prototype:

```c
typedef int Tk_ImageFileMatchProc(
    Tcl_Channel chan,
    const char *fileName,
    Tcl_Obj *format,
    int *widthPtr,
    int *heightPtr,
    Tcl_Interp *interp);
```

The fileName argument is the name of the file containing the image data, which is open for reading as chan. The format argument contains the value given for the -format option, or NULL if the option was not specified. If the data in the file appears to be in the format supported by this handler, the formatPtr->fileMatchProc procedure should store the width and height of the image in *widthPtr and *heightPtr respectively, and return 1. Otherwise it should return 0.
**STRINGMATCHPROC**

formatPtr->stringMatchProc provides the address of a procedure for Tk to call when it is searching for an image file format handler for suitable for reading data from a given string. formatPtr->stringMatchProc must match the following prototype:

```c
typedef int Tk_ImageStringMatchProc(
    Tcl_Obj *data,
    Tcl_Obj *format,
    int *widthPtr,
    int *heightPtr,
    Tcl_Interp *interp);
```

The data argument points to the object containing the image data. The format argument contains the value given for the -format option, or NULL if the option was not specified. If the data in the string appears to be in the format supported by this handler, the formatPtr->stringMatchProc procedure should store the width and height of the image in *widthPtr and *heightPtr respectively, and return 1. Otherwise it should return 0.

**FILEREADPROC**

formatPtr->fileReadProc provides the address of a procedure for Tk to call to read data from an image file into a photo image. formatPtr->fileReadProc must match the following prototype:

```c
typedef int Tk_ImageFileReadProc(
    Tcl_Interp *interp,
    Tcl_Channel chan,
    const char *fileName,
    Tcl_Obj *format,
    PhotoHandle imageHandle,
    int destX, int destY,
    ...);
```
The `interp` argument is the interpreter in which the command was invoked to read the image; it should be used for reporting errors. The image data is in the file named `fileName`, which is open for reading as `chan`. The `format` argument contains the value given for the `-format` option, or NULL if the option was not specified. The image data in the file, or a subimage of it, is to be read into the photo image identified by the handle `imageHandle`. The subimage of the data in the file is of dimensions `width x height` and has its top-left corner at coordinates `(srcX,srcY)`. It is to be stored in the photo image with its top-left corner at coordinates `(destX,destY)` using the `Tk_PhotoPutBlock` procedure. The return value is a standard Tcl return value.

**STRINGREADPROC**

`formatPtr->stringReadProc` provides the address of a procedure for Tk to call to read data from a string into a photo image. `formatPtr->stringReadProc` must match the following prototype:

```c
typedef int Tk_ImageStringReadProc(
    Tcl_Interp *interp,
    Tcl_Obj *data,
    Tcl_Obj *format,
    PhotoHandle imageHandle,
    int destX, int destY,
    int width, int height,
    int srcX, int srcY);
```

The `interp` argument is the interpreter in which the command was invoked to read the image; it should be used for reporting errors. The `data` argument points to the image data in object form. The `format` argument contains the value given for the `-format` option, or NULL if the option was not specified. The image data in the string, or a subimage of
it, is to be read into the photo image identified by the handle `imageHandle`. The subimage of the data in the string is of dimensions `width x height` and has its top-left corner at coordinates `(srcX,srcY)`. It is to be stored in the photo image with its top-left corner at coordinates `(destX,destY)` using the `Tk_PhotoPutBlock` procedure. The return value is a standard Tcl return value.

**FILEWRITEPROC**

`formatPtr->fileWriteProc` provides the address of a procedure for Tk to call to write data from a photo image to a file. `formatPtr->fileWriteProc` must match the following prototype:

```c
typedef int Tk_ImageFileWriteProc(
    Tcl_Interp *interp,
    const char *fileName,
    Tcl_Obj *format,
    Tk_PhotoImageBlock *blockPtr);
```

The `interp` argument is the interpreter in which the command was invoked to write the image; it should be used for reporting errors. The image data to be written are in memory and are described by the `Tk_PhotoImageBlock` structure pointed to by `blockPtr`; see the manual page FindPhoto(3) for details. The `fileName` argument points to the string giving the name of the file in which to write the image data. The `format` argument contains the value given for the `-format` option, or NULL if the option was not specified. The format string can contain extra characters after the name of the format. If appropriate, the `formatPtr->fileWriteProc` procedure may interpret these characters to specify further details about the image file. The return value is a standard Tcl return value.

**STRINGWRITEPROC**

`formatPtr->stringWriteProc` provides the address of a procedure for Tk to call to translate image data from a photo image into a string.
formatPtr->stringWriteProc must match the following prototype:

```c
typedef int Tk_ImageStringWriteProc(
    Tcl_Interp *interp,
    Tcl_Obj *format,
    Tk_PhotoImageBlock *blockPtr);
```

The `interp` argument is the interpreter in which the command was invoked to convert the image; it should be used for reporting errors. The image data to be converted are in memory and are described by the Tk_PhotoImageBlock structure pointed to by `blockPtr`; see the manual page FindPhoto(3) for details. The data for the string should be put in the interpreter `interp` result. The `format` argument contains the value given for the `-format` option, or NULL if the option was not specified. The format string can contain extra characters after the name of the format. If appropriate, the `formatPtr->stringWriteProc` procedure may interpret these characters to specify further details about the image file. The return value is a standard Tcl return value.

### LEGACY INTERFACE SUPPORT

In Tk 8.2 and earlier, the definition of all the function pointer types stored in fields of a Tk_PhotolImageFormat struct were incompatibly different. Legacy programs and libraries dating from those days may still contain code that defines extended Tk photo image formats using the old interface. The Tk header file will still support this legacy interface if the code is compiled with the macro `USE_OLD_IMAGE` defined. Alternatively, the legacy interfaces are used if the first character of `formatPtr->name` is an uppercase ASCII character (A-Z), and explicit casts are used to forgive the type mismatch. For example,

```c
static Tk_PhotolImageFormat myFormat = {
    "MyFormat",
    (Tk_ImageFileMatchProc *) FileMatch,
    NULL,
```
would define a minimal `Tk_PhotolImageFormat` that operates provide only file reading capability, where `FileMatch` and `FileRead` are written according to the legacy interfaces of Tk 8.2 or earlier.

Any stub-enabled extension providing an extended photo image format via the legacy interface enabled by the `USE_OLD_IMAGE` macro that is compiled against Tk 8.5 headers and linked against the Tk 8.5 stub library will produce a file that can be loaded only into interps with Tk 8.5 or later; that is, the normal stub-compatibility rules. If a developer needs to generate from such code a file that is loadable into interps with Tk 8.4 or earlier, they must use Tk 8.4 headers and stub libraries to do so.

Any new code written today should not make use of the legacy interfaces. Expect their support to go away in Tk 9.

**SEE ALSO**

`Tk_FindPhoto`, `Tk_PhotoPutBlock`

**KEYWORDS**

photo image, image file
Tk_CreateSelHandler, Tk_DeleteSelHandler - arrange to handle requests for a selection

**SYNOPSIS**

```c
#include <tk.h>

Tk_CreateSelHandler(tkwin, selection, target, proc, clientData, format)
Tk_DeleteSelHandler(tkwin, selection, target)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tk_Window tkwin (in)</td>
<td>Window for which <code>proc</code> will provide selection information.</td>
</tr>
<tr>
<td>Atom selection (in)</td>
<td>The name of the selection for which <code>proc</code> will provide selection information.</td>
</tr>
<tr>
<td>Atom target (in)</td>
<td>Form in which <code>proc</code> can provide the selection (e.g. STRING or FILE_NAME).</td>
</tr>
<tr>
<td></td>
<td>Corresponds to type arguments in <code>selection</code> commands.</td>
</tr>
<tr>
<td>Tk_SelectionProc *proc (in)</td>
<td>Procedure to invoke whenever the selection is owned by <code>tkwin</code> and the</td>
</tr>
</tbody>
</table>
selection contents are requested in the format given by target.

<table>
<thead>
<tr>
<th>ClientData</th>
<th>clientData (in)</th>
<th>Arbitrary one-word value to pass to proc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atom</td>
<td>format (in)</td>
<td>If the selection requestor is not in this process, format determines the representation used to transmit the selection to its requestor.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

**Tk_CreateSelHandler** arranges for a particular procedure (proc) to be called whenever selection is owned by tkwin and the selection contents are requested in the form given by target. Target should be one of the entries defined in the left column of Table 2 of the X Inter-Client Communication Conventions Manual (ICCCM) or any other form in which an application is willing to present the selection. The most common form is STRING.

Proc should have arguments and result that match the type **Tk_SelectionProc**:

```c
typedef int Tk_SelectionProc(
    ClientData clientData,
    int offset,
    char *buffer,
    int maxBytes);
```

The clientData parameter to proc is a copy of the clientData argument given to Tk_CreateSelHandler. Typically, clientData points to a data
structure containing application-specific information that is needed to retrieve the selection. Offset specifies an offset position into the selection, buffer specifies a location at which to copy information about the selection, and maxBytes specifies the amount of space available at buffer. Proc should place a NULL-terminated string at buffer containing maxBytes or fewer characters (not including the terminating NULL), and it should return a count of the number of non-NULL characters stored at buffer. If the selection no longer exists (e.g. it once existed but the user deleted the range of characters containing it), then proc should return -1.

When transferring large selections, Tk will break them up into smaller pieces (typically a few thousand bytes each) for more efficient transmission. It will do this by calling proc one or more times, using successively higher values of offset to retrieve successive portions of the selection. If proc returns a count less than maxBytes it means that the entire remainder of the selection has been returned. If proc's return value is maxBytes it means there may be additional information in the selection, so Tk must make another call to proc to retrieve the next portion.

Proc always returns selection information in the form of a character string. However, the ICCCM allows for information to be transmitted from the selection owner to the selection requestor in any of several formats, such as a string, an array of atoms, an array of integers, etc. The format argument to Tk_CreateSelHandler indicates what format should be used to transmit the selection to its requestor (see the middle column of Table 2 of the ICCCM for examples). If format is not STRING, then Tk will take the value returned by proc and divided it into fields separated by white space. If format is ATOM, then Tk will return the selection as an array of atoms, with each field in proc's result treated as the name of one atom. For any other value of format, Tk will return the selection as an array of 32-bit values where each field of proc's result is treated as a number and translated to a 32-bit value. In any event, the format atom is returned to the selection requestor along with the contents of the selection.

If Tk_CreateSelHandler is called when there already exists a handler
for *selection* and *target* on *tkwin*, then the existing handler is replaced with a new one.

**Tk_DeleteSelHandler** removes the handler given by *tkwin, selection*, and *target*, if such a handler exists. If there is no such handler then it has no effect.

**KEYWORDS**

format, handler, selection, target
NAME

Ttk_CreateTheme, Ttk_GetTheme, Ttk_GetDefaultTheme, Ttk_GetCurrentTheme - create and use Tk themes.

SYNOPSIS

Ttk_Theme Ttk_CreateTheme(interp, name, parentTheme);
Ttk_Theme Ttk_GetTheme(interp, name);
Ttk_Theme Ttk_GetDefaultTheme(interp);
Ttk_Theme Ttk_GetCurrentTheme(interp);

ARGUMENTS

Tcl_Interp * interp (in) The Tcl interpreter in which to register/query available themes.

Ttk_Theme parentTheme (in) Fallback or parent theme from which the new theme will inherit elements and layouts.

const char * name (in) The name of the theme.

DESCRIPTION

SEE ALSO

Ttk_RegisterLayout, Ttk_BuildLayout
NAME

Tk_GetHINSTANCE - retrieve the global application instance handle

SYNOPSIS

#include <tk.h>
HINSTANCE
Tk_GetHINSTANCE()

DESCRIPTION

Tk_GetHINSTANCE returns the Windows application instance handle for the Tk application. This function is only available on Windows platforms.

KEYWORDS

identifier, instance

Copyright © 1995-1997 Roger E. Critchlow Jr.
NAME

Tk_NameOfImage - Return name of image.

SYNOPSIS

#include <tk.h>
const char *
Tk_NameOfImage(typePtr)

ARGUMENTS

Tk_ImageMaster *masterPtr (in)  Token for image, which 
was passed to image 
manager's createProc 
when the image was 
created.

DESCRIPTION

This procedure is invoked by image managers to find out the name of 
an image. Given the token for the image, it returns the string name for 
the image.

KEYWORDS

image_manager, image_name

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
NAME
Tk_GetJoinStyle, Tk_NameOfJoinStyle - translate between strings and join styles

SYNOPSIS
#include <tk.h>

int

Tk_GetJoinStyle(interp, string, joinPtr)

const char *

Tk_NameOfJoinStyle(join)

ARGUMENTS

DESCRIPTION

KEYWORDS

NAME
Tk_GetJoinStyle, Tk_NameOfJoinStyle - translate between strings and join styles

SYNOPSIS
#include <tk.h>

int

Tk_GetJoinStyle(interp, string, joinPtr)

const char *

Tk_NameOfJoinStyle(join)

ARGUMENTS

Tcl_Interp *interp (in) Interpreter to use for error reporting.

const char *string (in) String containing name of
**DESCRIPTION**

**Tk_GetJoinStyle** places in \*joinPtr the X join style corresponding to \*string, which will be one of **JoinBevel**, **JoinMiter**, or **JoinRound**. Join styles are typically used in X graphics contexts to indicate how adjacent line segments should be joined together. See the X documentation for information on what each style implies.

Under normal circumstances the return value is **TCL_OK** and interp is unused. If \*string does not contain a valid join style or an abbreviation of one of these names, then an error message is stored in interp->result, **TCL_ERROR** is returned, and \*joinPtr is unmodified.

**Tk_NameOfJoinStyle** is the logical inverse of **Tk_GetJoinStyle**. Given a join style such as **JoinBevel** it returns a statically-allocated string corresponding to \*join. If \*join is not a legal join style, then “unknown join style” is returned.

**KEYWORDS**

bevel, join style, miter, round
NAME
Tk_GetJustifyFromObj, Tk_GetJustify, Tk_NameOfJustify - translate between strings and justification styles

SYNOPSIS
#include <tk.h>
int Tk_GetJustifyFromObj(interp, objPtr, justifyPtr)
int Tk_GetJustify(interp, string, justifyPtr)
const char * Tk_NameOfJustify(justify)

ARGUMENTS

DESCRIPTION
TK_JUSTIFY_LEFT
TK_JUSTIFY_RIGHT
TK_JUSTIFY_CENTER

KEYWORDS
### ARGUMENTS

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*<em>Tcl_Interp <em>interp</em></em> (in)</td>
<td>Interpreter to use for error reporting, or NULL.</td>
</tr>
<tr>
<td>*<em>Tcl_Obj <em>objPtr</em></em> (in/out)</td>
<td>String value contains name of justification style, one of “left”, “right”, or “center”. The internal rep will be modified to cache corresponding justify value.</td>
</tr>
<tr>
<td>*<em>const char <em>string</em></em> (in)</td>
<td>Same as <em>objPtr</em> except description of justification style is passed as a string.</td>
</tr>
<tr>
<td>*<em>int <em>justifyPtr</em></em> (out)</td>
<td>Pointer to location in which to store justify value corresponding to <em>objPtr</em> or <em>string</em>.</td>
</tr>
<tr>
<td><strong>Tk_Justify justify</strong> (in)</td>
<td>Justification style (one of the values listed below).</td>
</tr>
</tbody>
</table>

### DESCRIPTION

**Tk_GetJustifyFromObj** places in *justifyPtr* the justify value corresponding to *objPtr*’s value. This value will be one of the following:

**TK_JUSTIFY_LEFT**

Means that the text on each line should start at the left edge of the line; as a result, the right edges of lines may be ragged.

**TK_JUSTIFY_RIGHT**

Means that the text on each line should end at the right edge of the
line; as a result, the left edges of lines may be ragged.

**TK_JUSTIFY_CENTER**

Means that the text on each line should be centered; as a result, both the left and right edges of lines may be ragged.

Under normal circumstances the return value is **TCL_OK** and **interp** is unused. If **objPtr** does not contain a valid justification style or an abbreviation of one of these names, **TCL_ERROR** is returned, **justifyPtr** is unmodified, and an error message is stored in **interp**'s result if **interp** is not NULL. **Tk_GetJustifyFromObj** caches information about the return value in **objPtr**, which speeds up future calls to **Tk_GetJustifyFromObj** with the same **objPtr**.

**Tk_GetJustify** is identical to **Tk_GetJustifyFromObj** except that the description of the justification is specified with a string instead of an object. This prevents **Tk_GetJustify** from caching the return value, so **Tk_GetJustify** is less efficient than **Tk_GetJustifyFromObj**.

**Tk_NameOfJustify** is the logical inverse of **Tk_GetJustify**. Given a justify value it returns a statically-allocated string corresponding to **justify**. If **justify** is not a legal justify value, then “unknown justification style” is returned.

**KEYWORDS**

center, fill, justification, string

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1994 The Regents of the University of California.
Copyright © 1994-1998 Sun Microsystems, Inc.
 Tk_OwnSelection - make a window the owner of the primary selection

SYNOPSIS

#include <tk.h>
Tk_OwnSelection(tkwin, selection, proc, clientData)

ARGUMENTS

Tk_Window tkwin (in)  Window that is to become new selection owner.
Atom selection (in)  The name of the selection to be owned, such as XA_PRIMARY.
Tk_LostSelProc *proc (in)  Procedure to invoke when tkwin loses selection ownership later.
ClientData clientData (in)  Arbitrary one-word value to pass to proc.

DESCRIPTION

Tk_OwnSelection arranges for tkwin to become the new owner of the selection specified by the atom selection. After this call completes, future requests for the selection will be directed to handlers created for tkwin using Tk_CreateSelHandler. When tkwin eventually loses the
selection ownership, proc will be invoked so that the window can clean itself up (e.g. by unhighlighting the selection). Proc should have arguments and result that match the type Tk_LostSelProc:

```c
typedef void Tk_LostSelProc(ClientData clientData);
```

The clientData parameter to proc is a copy of the clientData argument given to Tk_OwnSelection, and is usually a pointer to a data structure containing application-specific information about tkwin.

KEYWORDS

own, selection owner
NAME

Tk_ParseArgv - process command-line options

SYNOPSIS

#include <tk.h>

int
Tk_ParseArgv(interp, tkwin, argcPtr, argv, argTable, flags)

ARGUMENTS

DESCRIPTION

NAME

SYNOPSIS

#include <tk.h>

int
Tk_ParseArgv(interp, tkwin, argcPtr, argv, argTable, flags)

ARGUMENTS

DESCRIPTION

NAME
Tk_ParseArgv - process command-line options

**SYNOPSIS**

```c
#include <tk.h>
int Tk_ParseArgv(Tcl_Interp *interp, Tk_Window tkwin, int argcPtr, const char **argv, Tk_ArgvInfo *argTable, int flags)
```

**ARGUMENTS**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Tcl_Interp *interp</code> (in)</td>
<td>Interpreter to use for returning error messages.</td>
</tr>
<tr>
<td><code>Tk_Window tkwin</code> (in)</td>
<td>Window to use when arguments specify Tk options. If NULL, then no Tk options will be processed.</td>
</tr>
<tr>
<td><code>int argcPtr</code> (in/out)</td>
<td>Pointer to number of arguments in argv; gets modified to hold number of unprocessed arguments that remain after the call.</td>
</tr>
<tr>
<td><code>const char **argv</code> (in/out)</td>
<td>Command line arguments passed to main program. Modified to hold unprocessed arguments that remain after the call.</td>
</tr>
<tr>
<td><code>Tk_ArgvInfo *argTable</code> (in)</td>
<td>Array of argument descriptors, terminated by element with type <code>TK_ARGV_END</code>.</td>
</tr>
<tr>
<td><code>int flags</code> (in)</td>
<td>If non-zero, then it</td>
</tr>
</tbody>
</table>
specifies one or more flags that control the parsing of arguments. Different flags may be OR’ed together. The flags currently defined are

- **TK_ARGV_DONT_SKIP_FIRST_ARG**
- **TK_ARGV_NO_ABBREV**
- **TK_ARGV_NO_LEFTOVERS**
- **TK_ARGV_NO_DEFAULTS**

### DESCRIPTION

**Tk_ParseArgv** processes an array of command-line arguments according to a table describing the kinds of arguments that are expected. Each of the arguments in `argv` is processed in turn: if it matches one of the entries in `argTable`, the argument is processed according to that entry and discarded. The arguments that do not match anything in `argTable` are copied down to the beginning of `argv` (retaining their original order) and returned to the caller. At the end of the call **Tk_ParseArgv** sets `*argcPtr` to hold the number of arguments that are left in `argv`, and `argv[*argcPtr]` will hold the value NULL. Normally, **Tk_ParseArgv** assumes that `argv[0]` is a command name, so it is treated like an argument that does not match `argTable` and returned to the caller; however, if the **TK_ARGV_DONT_SKIP_FIRST_ARG** bit is set in `flags` then `argv[0]` will be processed just like the other elements of `argv`.

**Tk_ParseArgv** normally returns the value **TCL_OK**. If an error occurs while parsing the arguments, then **TCL_ERROR** is returned and **Tk_ParseArgv** will leave an error message in `interp->result` in the standard Tcl fashion. In the event of an error return, `*argvPtr` will not have been modified, but `argv` could have been partially modified. The possible causes of errors are explained below.

The **argTable** array specifies the kinds of arguments that are expected;
each of its entries has the following structure:

```c
typedef struct {
    char *key;
    int type;
    char *src;
    char *dst;
    char *help;
} Tk_ArgvInfo;
```

The `key` field is a string such as “-display” or “-bg” that is compared with the values in `argv`. `Type` indicates how to process an argument that matches `key` (more on this below). `Src` and `dst` are additional values used in processing the argument. Their exact usage depends on `type`, but typically `src` indicates a value and `dst` indicates where to store the value. The `char *` declarations for `src` and `dst` are placeholders: the actual types may be different. Lastly, `help` is a string giving a brief description of this option; this string is printed when users ask for help about command-line options.

When processing an argument in `argv`, `Tk_ParseArgv` compares the argument to each of the `key`'s in `argTable`. `Tk_ParseArgv` selects the first specifier whose `key` matches the argument exactly, if such a specifier exists. Otherwise `Tk_ParseArgv` selects a specifier for which the argument is a unique abbreviation. If the argument is a unique abbreviation for more than one specifier, then an error is returned. If there is no matching entry in `argTable`, then the argument is skipped and returned to the caller.

Once a matching argument specifier is found, `Tk_ParseArgv` processes the argument according to the `type` field of the specifier. The argument that matched `key` is called “the matching argument” in the descriptions below. As part of the processing, `Tk_ParseArgv` may also use the next argument in `argv` after the matching argument, which is called “the following argument”. The legal values for `type`, and the processing that they cause, are as follows:
**TK_ARGV_END**
Marks the end of the table. The last entry in `argTable` must have this type; all of its other fields are ignored and it will never match any arguments.

**TK_ARGV_CONSTANT**
`Src` is treated as an integer and `dst` is treated as a pointer to an integer. `Src` is stored at `*dst`. The matching argument is discarded.

**TK_ARGV_INT**
The following argument must contain an integer string in the format accepted by `strtol` (e.g. “0” and “0x” prefixes may be used to specify octal or hexadecimal numbers, respectively). `Dst` is treated as a pointer to an integer; the following argument is converted to an integer value and stored at `*dst`. `Src` is ignored. The matching and following arguments are discarded from `argv`.

**TK_ARGV_FLOAT**
The following argument must contain a floating-point number in the format accepted by `strtol`. `Dst` is treated as the address of a double-precision floating point value; the following argument is converted to a double-precision value and stored at `*dst`. The matching and following arguments are discarded from `argv`.

**TK_ARGV_STRING**
In this form, `dst` is treated as a pointer to a (char *); `Tk_ParseArgv` stores at `*dst` a pointer to the following argument, and discards the matching and following arguments from `argv`. `Src` is ignored.

**TK_ARGV_UID**
This form is similar to **TK_ARGV_STRING**, except that the argument is turned into a `Tk_Uid` by calling `Tk_GetUid`. `Dst` is treated as a pointer to a `Tk_Uid`; `Tk_ParseArgv` stores at `*dst` the `Tk_Uid` corresponding to the following argument, and discards the matching and following arguments from `argv`. `Src` is ignored.

**TK_ARGV_CONST_OPTION**
This form causes a Tk option to be set (as if the `option` command
had been invoked). The src field is treated as a pointer to a string giving the value of an option, and dst is treated as a pointer to the name of the option. The matching argument is discarded. If tkwin is NULL, then argument specifiers of this type are ignored (as if they did not exist).

**TK_ARGV_OPTION_VALUE**
This form is similar to **TK_ARGVCONST_OPTION**, except that the value of the option is taken from the following argument instead of from src. Dst is used as the name of the option. Src is ignored. The matching and following arguments are discarded. If tkwin is NULL, then argument specifiers of this type are ignored (as if they did not exist).

**TK_ARGV_OPTION_NAME_VALUE**
In this case the following argument is taken as the name of a Tk option and the argument after that is taken as the value for that option. Both src and dst are ignored. All three arguments are discarded from argv. If tkwin is NULL, then argument specifiers of this type are ignored (as if they did not exist).

**TK_ARGV_HELP**
When this kind of option is encountered, Tk_ParseArgv uses the help fields of argTable to format a message describing all the valid arguments. The message is placed in interp->result and Tk_ParseArgv returns TCL_ERROR. When this happens, the caller normally prints the help message and aborts. If the key field of a **TK_ARGV_HELP** specifier is NULL, then the specifier will never match any arguments; in this case the specifier simply provides extra documentation, which will be included when some other **TK_ARGV_HELP** entry causes help information to be returned.

**TK_ARGV_REST**
This option is used by programs or commands that allow the last several of their options to be the name and/or options for some other program. If a **TK_ARGV_REST** argument is found, then Tk_ParseArgv does not process any of the remaining arguments;
it returns them all at the beginning of *argv (along with any other unprocessed arguments). In addition, Tk_ParseArgv treats *dst as the address of an integer value, and stores at *dst the index of the first of the TK_ARGV_REST options in the returned *argv. This allows the program to distinguish the TK_ARGV_REST options from other unprocessed options that preceded the TK_ARGV_REST.

TK_ARGV_FUNC
For this kind of argument, *src is treated as the address of a procedure, which is invoked to process the following argument. The procedure should have the following structure:

```c
int func(dst, key, nextArg)
    char *dst;
    char *key;
    char *nextArg;
{
}
```

The *dst and *key parameters will contain the corresponding fields from the *argTable entry, and *nextArg will point to the following argument from *argv (or NULL if there are not any more arguments left in *argv). If func uses *nextArg (so that Tk_ParseArgv should discard it), then it should return 1. Otherwise it should return 0 and TkParseArgv will process the following argument in the normal fashion. In either event the matching argument is discarded.

TK_ARGV_GENFUNC
This form provides a more general procedural escape. It treats *src as the address of a procedure, and passes that procedure all of the remaining arguments. The procedure should have the following form:
The `dst` and `key` parameters will contain the corresponding fields from the `argTable` entry. `Interp` will be the same as the `interp` argument to `Tcl_ParseArgv`. `Argc` and `argv` refer to all of the options after the matching one. `Genfunc` should behave in a fashion similar to `Tk_ParseArgv`: parse as many of the remaining arguments as it can, then return any that are left by compacting them to the beginning of `argv` (starting at `argv[0]`). `Genfunc` should return a count of how many arguments are left in `argv`; `Tk_ParseArgv` will process them. If `genfunc` encounters an error then it should leave an error message in `interp->result`, in the usual Tcl fashion, and return -1; when this happens `Tk_ParseArgv` will abort its processing and return `TCL_ERROR`.

### FLAGS

**TK_ARGV_DONT_SKIP_FIRST_ARG**

`Tk_ParseArgv` normally treats `argv[0]` as a program or command name, and returns it to the caller just as if it had not matched `argTable`. If this flag is given, then `argv[0]` is not given special treatment.

**TK_ARGV_NO_ABBREV**

Normally, `Tk_ParseArgv` accepts unique abbreviations for `key` values in `argTable`. If this flag is given then only exact matches will be acceptable.
TK_ARGV_NO_LEFTOVERS
Normally, Tk_ParseArgv returns unrecognized arguments to the caller. If this bit is set in flags then Tk_ParseArgv will return an error if it encounters any argument that does not match argTable. The only exception to this rule is argv[0], which will be returned to the caller with no errors as long as TK_ARGV_DONT_SKIP_FIRST_ARG is not specified.

TK_ARGV_NO_DEFAULTS
Normally, Tk_ParseArgv searches an internal table of standard argument specifiers in addition to argTable. If this bit is set in flags, then Tk_ParseArgv will use only argTable and not its default table.

EXAMPLE
Here is an example definition of an argTable and some sample command lines that use the options. Note the effect on argc and argv; arguments processed by Tk_ParseArgv are eliminated from argv, and argc is updated to reflect reduced number of arguments.

/*
 * Define and set default values for globals.
 */
int debugFlag = 0;
int numReps = 100;
char defaultFileName[] = "out";
char *fileName = defaultFileName;
Boolean exec = FALSE;

/*
 * Define option descriptions.
 */
Tk_ArgvInfo argTable[] = {
    {"-X", TK_ARGV_CONSTANT, (char *) 1, (char *) &debugFlag, "Turn on debugging printfs"},
    {"-N", TK_ARGV_INT, (char *) NULL, (char *) &numReps, "Number of repetitions"},
main(argc, argv)
    int argc;
    char *argv[];
{
    ...

    if (Tk_ParseArgv(interp, tkwin, &argc, argv, argTable, 0) != TCL_OK)
    {
        fprintf(stderr, "%s\n", interp->result);
        exit(1);
    }

    /*
     * Remainder of the program.
     */
}

Note that default values can be assigned to variables named in
argTable: the variables will only be overwritten if the particular
arguments are present in argv. Here are some example command lines
and their effects.

    prog -N 200 infile       # just sets the numReps variable
    prog -of out200 infile   # sets fileName to reference "out200"
    prog -XN 10 infile      # sets the debug flag, also

In all of the above examples, argc will be set by Tk_ParseArgv to 2,
argv[0] will be “prog”, argv[1] will be “infilename”, and argv[2] will be NULL.

**KEYWORDS**

arguments, command line, options

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990-1992 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
**NAME**

Tk_GetOption - retrieve an option from the option database

**SYNOPSIS**

```c
#include <tk.h>
Tk_Uid
Tk_GetOption(tkwin, name, class)
```

**ARGUMENTS**

- **Tk_Window tkwin (in)**: Token for window.
- **const char *name (in)**: Name of desired option.
- **const char *class (in)**: Class of desired option. Null means there is no class for this option; do lookup based on name only.

**DESCRIPTION**

This procedure is invoked to retrieve an option from the database associated with `tkwin`'s main window. If there is an option for `tkwin` that matches the given `name` or `class`, then it is returned in the form of a `Tk_Uid`. If multiple options match `name` and `class`, then the highest-priority one is returned. If no option matches, then NULL is returned.

`Tk_GetOption` caches options related to `tkwin` so that successive calls
for the same *tkwin* will execute much more quickly than successive calls for different windows.

**KEYWORDS**

*class, name, option, retrieve*

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1990 The Regents of the University of California.
Copyright © 1994-1996 Sun Microsystems, Inc.
NAME

Tk_DeleteImage - Destroy an image.

SYNOPSIS

#include <tk.h>
Tk_DeleteImage(interp, name)

ARGUMENTS

Tcl_Interp *interp (in)  
Interpreter for which the image was created.

const char *name (in)  
Name of the image.

DESCRIPTION

Tk_DeleteImage deletes the image given by interp and name, if there is one. All instances of that image will redisplay as empty regions. If the given image does not exist then the procedure has no effect.

KEYWORDS

delete image, image manager

Copyright © 1995-1997 Roger E. Critchlow Jr.
Copyright © 1995-1996 Sun Microsystems, Inc.
<p>| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
| abort | break, Panic |
| above | Restack |
| absolute file name | filename |
| access | Access, FileSystem |
| access mode | open |
| access point | OpenFileChnl |
| access position | seek, tell |
| add | incr, AddOption |
| alias | interp, safe, loadTk, CrtSlave |
| alloc | Alloc |
| allocation | Alloc |
| alpha | GetVersion |
| anchor | ConfigWidg, SetOptions |
| anchor position | GetAnchor |
| anonymous function | apply |
| appearance | ttk_image, ttk_style, ttk_vsapi |
| append | append, lappend, open, clipboard, DString, ListObj, SetResult, StringObj, Clipboard |
| application | dde, destroy, send, AppInit, Init, MainLoop, MainWin, SetAppName |</p>
<table>
<thead>
<tr>
<th>Concept</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>application name</td>
<td>tk</td>
</tr>
<tr>
<td>application-specific initialization</td>
<td>SourceRCFile, Tcl_Main, Tk_Main</td>
</tr>
<tr>
<td>architecture</td>
<td>platform, platform_shell</td>
</tr>
<tr>
<td>argument</td>
<td>tclsh, apply, proc, AppInit</td>
</tr>
<tr>
<td>arguments</td>
<td>ParseArgv</td>
</tr>
<tr>
<td>arithmetic</td>
<td>expr, tclvars</td>
</tr>
<tr>
<td>array</td>
<td>array, SetVar</td>
</tr>
<tr>
<td>aspect ratio</td>
<td>wm</td>
</tr>
<tr>
<td>assign</td>
<td>lassign</td>
</tr>
<tr>
<td>association</td>
<td>AssocData</td>
</tr>
<tr>
<td>asynchronous event</td>
<td>Async</td>
</tr>
<tr>
<td>asynchronous I/O</td>
<td>fileevent</td>
</tr>
<tr>
<td>atom</td>
<td>winfo, GetUid, InternAtom</td>
</tr>
<tr>
<td>attributes</td>
<td>file, ConfigWind, WindowId</td>
</tr>
<tr>
<td>auto-exec</td>
<td>library</td>
</tr>
<tr>
<td>auto-load</td>
<td>library, packagens, pkgMkIndex</td>
</tr>
<tr>
<td>auto-loading</td>
<td>safe, loadTk</td>
</tr>
<tr>
<td>auto_mkindex</td>
<td>safe, loadTk</td>
</tr>
<tr>
<td>background</td>
<td>BackgdErr, DetachPids, 3DBorder</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>background error</td>
<td>bgerror, tkerror</td>
</tr>
<tr>
<td>backlash</td>
<td>Backslash, SplitList, Utf</td>
</tr>
<tr>
<td>backlash substitution</td>
<td>subst, ParseCmd, SubstObj</td>
</tr>
<tr>
<td>beep</td>
<td>bell</td>
</tr>
<tr>
<td>bell</td>
<td>bell</td>
</tr>
<tr>
<td>below</td>
<td>Restack</td>
</tr>
<tr>
<td>beta</td>
<td>GetVersion</td>
</tr>
<tr>
<td>bevel</td>
<td>GetJoinStl</td>
</tr>
<tr>
<td>bgerror</td>
<td>BackgdErr</td>
</tr>
<tr>
<td>binary</td>
<td>binary, fconfigure, FindExec</td>
</tr>
<tr>
<td>binary code</td>
<td>load, unload</td>
</tr>
<tr>
<td>binary library</td>
<td>RegConfig</td>
</tr>
<tr>
<td>bind</td>
<td>socket, keysyms, CrtCommand, CrtObjCmd, CrtCmHdlr, CrtGenHdlr, EventHndlr</td>
</tr>
<tr>
<td>binding</td>
<td>bind, bindtags, event, keysyms, BindTable</td>
</tr>
<tr>
<td>bisque</td>
<td>palette</td>
</tr>
<tr>
<td>bitmap</td>
<td>bitmap, dialog, CanvPsY, ConfigWidg, GetBitmap,</td>
</tr>
<tr>
<td>blocking</td>
<td>SetOptions, close, fblocked, fconfigure, fcopy, fileevent, flush, gets, read, CrtChannel, CrtChnlHdlr, OpenFileChnl</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>boolean</td>
<td>expr, if, BoolObj, ExprLong, ExprLongObj, getInt, LinkVar, ConfigWidg, SetOptions</td>
</tr>
<tr>
<td>boolean value</td>
<td>while</td>
</tr>
<tr>
<td>border</td>
<td>3DBorder, ConfigWidg, ConfigWind, SetOptions</td>
</tr>
<tr>
<td>box</td>
<td>ttk_Geometry</td>
</tr>
<tr>
<td>braces</td>
<td>ParseCmd</td>
</tr>
<tr>
<td>break</td>
<td>break, return, AllowExc</td>
</tr>
<tr>
<td>buffer</td>
<td>flush</td>
</tr>
<tr>
<td>buffered I/O</td>
<td>OpenFileChnl</td>
</tr>
<tr>
<td>buffering</td>
<td>fconfigure</td>
</tr>
<tr>
<td>butt</td>
<td>GetCapStyl</td>
</tr>
<tr>
<td>butt</td>
<td>button, ttk_button, ttk_checkbutton, ttk_menubutton, ttk_radiobutton</td>
</tr>
<tr>
<td>byte array</td>
<td>fconfigure, ByteArrObj</td>
</tr>
<tr>
<td>bytecode</td>
<td>tclvars</td>
</tr>
<tr>
<td>cache</td>
<td>callback</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td>InternAtom</td>
</tr>
<tr>
<td>channel closing</td>
<td>CrtCloseHdr</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>channel driver</td>
<td>CrtChannel,</td>
</tr>
<tr>
<td></td>
<td>OpenFileChnl,</td>
</tr>
<tr>
<td></td>
<td>SetChanErr</td>
</tr>
<tr>
<td>channel registration</td>
<td>CrtChannel</td>
</tr>
<tr>
<td>channel type</td>
<td>CrtChannel,</td>
</tr>
<tr>
<td></td>
<td>SetChanErr</td>
</tr>
<tr>
<td>character</td>
<td>string</td>
</tr>
<tr>
<td>check</td>
<td>ttk_checkbutton</td>
</tr>
<tr>
<td>checkbutton</td>
<td>checkbutton</td>
</tr>
<tr>
<td>child</td>
<td>DetachPids</td>
</tr>
<tr>
<td>children</td>
<td>winfo</td>
</tr>
<tr>
<td>choice</td>
<td>ttk_combobox</td>
</tr>
<tr>
<td>class</td>
<td>options, winfo,</td>
</tr>
<tr>
<td></td>
<td>AddOption,</td>
</tr>
<tr>
<td></td>
<td>GetOption,</td>
</tr>
<tr>
<td></td>
<td>SetClass,</td>
</tr>
<tr>
<td></td>
<td>SetClassProcs</td>
</tr>
<tr>
<td>classification</td>
<td>UniCharIsAlpha</td>
</tr>
<tr>
<td>cleanup</td>
<td>Exit</td>
</tr>
<tr>
<td>clear</td>
<td>clipboard,</td>
</tr>
<tr>
<td></td>
<td>selection,</td>
</tr>
<tr>
<td></td>
<td>Clipboard,</td>
</tr>
<tr>
<td></td>
<td>ClrSelect</td>
</tr>
<tr>
<td>client</td>
<td>OpenTcp</td>
</tr>
<tr>
<td>clientData</td>
<td>TraceCmd, TraceVar</td>
</tr>
<tr>
<td>clipboard</td>
<td>clipboard,</td>
</tr>
<tr>
<td></td>
<td>Clipboard</td>
</tr>
<tr>
<td>clock</td>
<td>clock, CrtTimerHdr,</td>
</tr>
<tr>
<td></td>
<td>QWinEvent</td>
</tr>
<tr>
<td>close</td>
<td>close</td>
</tr>
<tr>
<td>color</td>
<td>colors, palette,</td>
</tr>
<tr>
<td></td>
<td>photo, 3DBorder,</td>
</tr>
<tr>
<td></td>
<td>CanvPsY, ConfigWidg,</td>
</tr>
<tr>
<td></td>
<td>ConfigWind,</td>
</tr>
<tr>
<td></td>
<td>GetColor, SetOptions</td>
</tr>
<tr>
<td>color selection dialog</td>
<td>chooseColor</td>
</tr>
<tr>
<td>colormap</td>
<td>GetClrmap, GetVisual, SetVisual, WindowId</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>command</td>
<td>info, mathop, namespace, rename, trace, ttk_button, ApplInit, CrtCommand, CrtInterp, CrtObjCmd, CrtSlave, CrtTrace, Namespace, ParseCmd, RecEvalObj, RecordEval, SetResult, TraceCmd, WrongNumArgs</td>
</tr>
<tr>
<td>command line</td>
<td>ParseArgv</td>
</tr>
<tr>
<td>command substitution</td>
<td>subst, SubstObj</td>
</tr>
<tr>
<td>command-line arguments</td>
<td>Tcl_Main, Tk_Main</td>
</tr>
<tr>
<td>commands</td>
<td>Limit</td>
</tr>
<tr>
<td>compare</td>
<td>expr, string</td>
</tr>
<tr>
<td>compiler</td>
<td>tclvars</td>
</tr>
<tr>
<td>complete command</td>
<td>CmdCmplt</td>
</tr>
<tr>
<td>compression</td>
<td>ChnlStack</td>
</tr>
<tr>
<td>concat</td>
<td>Obj</td>
</tr>
<tr>
<td>concatenate</td>
<td>concat, eval, Concat, StringObj</td>
</tr>
<tr>
<td>condition variable</td>
<td>Thread</td>
</tr>
<tr>
<td>conditional</td>
<td>if</td>
</tr>
<tr>
<td>configuration</td>
<td>RegConfig</td>
</tr>
<tr>
<td>configuration option</td>
<td>SetOptions</td>
</tr>
<tr>
<td>configuration options</td>
<td>ConfigWind</td>
</tr>
<tr>
<td>configure</td>
<td>ttk_widget, ConfigWind</td>
</tr>
<tr>
<td>connection</td>
<td>socket</td>
</tr>
<tr>
<td>console</td>
<td>console, CrtConsoleChan</td>
</tr>
<tr>
<td>container</td>
<td>ttk_frame, ttk_labelframe</td>
</tr>
<tr>
<td>containing</td>
<td>CoordToWin</td>
</tr>
<tr>
<td>context</td>
<td>uplevel, upvar</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>continue</td>
<td>continue, return, AllowExc</td>
</tr>
<tr>
<td>conversion</td>
<td>GetInt, PrintDb, GetDash</td>
</tr>
<tr>
<td>conversion specifier</td>
<td>format, scan</td>
</tr>
<tr>
<td>convert</td>
<td>Encoding, SplitList, GetPixels</td>
</tr>
<tr>
<td>coordinates</td>
<td>CoordToWin, GetRootCrd</td>
</tr>
<tr>
<td>copy files</td>
<td>file</td>
</tr>
<tr>
<td>cpu architecture</td>
<td>platform, platform_shell</td>
</tr>
<tr>
<td>create</td>
<td>dict, open, CrtCommand, CrtInterp, CrtObjCmd, CrtTrace, CrtWindow</td>
</tr>
<tr>
<td>ctype</td>
<td>string</td>
</tr>
<tr>
<td>current directory</td>
<td>filename</td>
</tr>
<tr>
<td>cursor</td>
<td>cursors, ConfigWidg, GetCursor, SetCaret, SetOptions</td>
</tr>
<tr>
<td>custom</td>
<td>ConfigWidg</td>
</tr>
<tr>
<td><strong>dash</strong></td>
<td>GetDash</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>database</strong></td>
<td>option</td>
</tr>
<tr>
<td><strong>dde</strong></td>
<td>dde, send</td>
</tr>
<tr>
<td><strong>default</strong></td>
<td>ttk_button</td>
</tr>
<tr>
<td><strong>deferred creation</strong></td>
<td>CrtWindow</td>
</tr>
<tr>
<td><strong>deiconify</strong></td>
<td>wm</td>
</tr>
<tr>
<td><strong>delete</strong></td>
<td>rename, CallDel, CrtCommand, CrtInterp, CrtObjCmd, CrtTrace</td>
</tr>
<tr>
<td><strong>delete files</strong></td>
<td>file</td>
</tr>
<tr>
<td><strong>deletion procedure</strong></td>
<td>AssocData</td>
</tr>
<tr>
<td><strong>depressed</strong></td>
<td>3DBorder</td>
</tr>
<tr>
<td><strong>destroy</strong></td>
<td>destroy, CrtWindow</td>
</tr>
<tr>
<td><strong>detach</strong></td>
<td>DetachPids</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>dialog</td>
<td><code>chooseDirectory</code>, <code>dialog</code></td>
</tr>
<tr>
<td>dict</td>
<td><code>DictObj</code></td>
</tr>
<tr>
<td>dict object</td>
<td><code>DictObj</code></td>
</tr>
<tr>
<td>dictionary</td>
<td><code>dict</code>, <code>DictObj</code></td>
</tr>
<tr>
<td>dictionary object</td>
<td><code>DictObj</code></td>
</tr>
<tr>
<td>directory</td>
<td><code>file</code>, <code>chooseDirectory</code></td>
</tr>
<tr>
<td>display</td>
<td><code>CrtWindow</code>, <code>InternAtom</code>, <code>WindowId</code></td>
</tr>
<tr>
<td>domain name</td>
<td><code>socket</code></td>
</tr>
<tr>
<td>double</td>
<td><code>DoubleObj</code>, <code>ExprLong</code>, <code>ExprLongObj</code>, <code>GetInt</code>, <code>ConfigWidg</code>, <code>SetOptions</code></td>
</tr>
<tr>
<td>double object</td>
<td><code>DoubleObj</code></td>
</tr>
<tr>
<td>double type</td>
<td><code>DoubleObj</code></td>
</tr>
<tr>
<td>double-precision</td>
<td><code>PrintDbl</code></td>
</tr>
<tr>
<td>dynamic loading</td>
<td><code>Exit</code></td>
</tr>
<tr>
<td>dynamic string</td>
<td><code>DString</code></td>
</tr>
</tbody>
</table>

Copyright © 1989-1994 The Regents of the University of California
Copyright © 1992-1999 Karl Lehenbauer & Mark Diekhans
Copyright © 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright © 1994 The Australian National University
Copyright © 1994-2000 Sun Microsystems, Inc
Copyright © 1995-1997 Roger E. Critchlow Jr
Copyright © 1997-2000 Ajuba Solutions
Copyright © 1997-2000 Scriptics Corporation
Copyright © 1998 Mark Harrison
Copyright © 2000 Jeffrey Hobbs
Copyright © 2001 ActiveState Tool Corp
Copyright © 2001 Vincent Darley
Copyright © 2001-2004 ActiveState Corporation
Copyright © 2001-2005 Kevin B. Kenny <kennykb(at)acm.org>
Copyright © 2001-2008 Donal K. Fellows
Copyright © 2002-2008 Andreas Kupries <andreas_kupries(at)users.sourceforge.net>
Copyright © 2003 George Petasis <petasis(at)jil.demokritos.gr>
Copyright © 2003 Simon Geard
Copyright © 2003-2006 Joe English
Copyright © 2006 Miguel Sofer
Copyright © 2006-2007 Daniel A. Steffen <das(at)users.sourceforge.net>
Copyright © 2006-2008 ActiveState Software Inc
Copyright © 2008 Pat Thoyts
element

```
array
```

```
element names
```

```
join, lappend, lassign,
```

```
encoding
```

```
TagConfig
```

```
eof, fcopy, gets, read,
```

```
end application
```

```
Exit
```

```
end of file
```

```
eof, fcopy, gets, read,
```

```
environment
```

```
tclvars, Environment
```

```
equal
```

```
string
```

```
erro
```

```
SetErrno
```

```
error
```

```
catch, error, return,
tclvars, unknown,
AddErrInfo, BackgdErr,
Panic, CrtErrHdlr
```
<table>
<thead>
<tr>
<th>error code</th>
<th>SetErrno</th>
</tr>
</thead>
<tbody>
<tr>
<td>error message</td>
<td>WrongNumArgs</td>
</tr>
<tr>
<td>error messages</td>
<td>SetChanErr</td>
</tr>
<tr>
<td>evaluate</td>
<td>eval, ExprLong,</td>
</tr>
<tr>
<td></td>
<td>ExprLongObj</td>
</tr>
<tr>
<td>event</td>
<td>history, update,</td>
</tr>
<tr>
<td></td>
<td>ywait, bind,</td>
</tr>
<tr>
<td></td>
<td>bindtags, event,</td>
</tr>
<tr>
<td></td>
<td>DoOneEvent, Notifer,</td>
</tr>
<tr>
<td></td>
<td>RecEvalObj, RecordEval,</td>
</tr>
<tr>
<td></td>
<td>BindTable, CrtCmHdlr,</td>
</tr>
<tr>
<td></td>
<td>CrtErrHdlr, CrtGenHdlr,</td>
</tr>
<tr>
<td></td>
<td>EventHndlr, HandleEvent,</td>
</tr>
<tr>
<td></td>
<td>MainLoop, RestrictEv</td>
</tr>
<tr>
<td>event handler</td>
<td>fileevent</td>
</tr>
<tr>
<td>event queue</td>
<td>Notifier</td>
</tr>
<tr>
<td>event sources</td>
<td>Notifier</td>
</tr>
<tr>
<td>events</td>
<td>chan, focus, CrtCmHdlr,</td>
</tr>
<tr>
<td></td>
<td>QWinEvent</td>
</tr>
<tr>
<td>exception</td>
<td>AllowExc</td>
</tr>
<tr>
<td>executable file</td>
<td>FindExec</td>
</tr>
<tr>
<td>execute</td>
<td>exec, Eval, RecEvalObj,</td>
</tr>
<tr>
<td></td>
<td>RecordEval</td>
</tr>
<tr>
<td>exist</td>
<td>glob</td>
</tr>
<tr>
<td>exit</td>
<td>exit, Exit</td>
</tr>
<tr>
<td>exported</td>
<td>namespace</td>
</tr>
<tr>
<td>exposed commands</td>
<td>CrtSlave</td>
</tr>
<tr>
<td>expression</td>
<td>expr, mathop, CrtMathFnc,</td>
</tr>
<tr>
<td></td>
<td>ExprLong, ExprLongObj,</td>
</tr>
<tr>
<td></td>
<td>ParseCmd</td>
</tr>
<tr>
<td>false</td>
<td>if</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>fatal</td>
<td>Panic</td>
</tr>
<tr>
<td>file</td>
<td>file, glob, open, pid, seek, source, CrtFileHdlr, Eval, SplitPath</td>
</tr>
<tr>
<td>file events</td>
<td>Notifier</td>
</tr>
<tr>
<td>file handle</td>
<td>GetOpenFl</td>
</tr>
<tr>
<td>file name</td>
<td>Translate</td>
</tr>
<tr>
<td>file selection dialog</td>
<td>getOpenFile</td>
</tr>
<tr>
<td>filename</td>
<td>SplitPath</td>
</tr>
<tr>
<td>filesystem</td>
<td>FileSystem</td>
</tr>
<tr>
<td>fill</td>
<td>GetJustify</td>
</tr>
<tr>
<td>filter</td>
<td>dict, fconfigure, RestrictEv</td>
</tr>
<tr>
<td>floating-point</td>
<td>GetInt, PrintDbl</td>
</tr>
<tr>
<td>flush</td>
<td>flush, update, OpenFileChnl</td>
</tr>
<tr>
<td>flushing</td>
<td>fconfigure</td>
</tr>
<tr>
<td>focus</td>
<td>focus, focusNext, CanvTkwin, CanvTxtInfo, CrtItemType, DrawFocHlt</td>
</tr>
<tr>
<td>focus model</td>
<td>wm</td>
</tr>
<tr>
<td>font</td>
<td>font, CanvPsY, ConfigWidg, FontId, GetFont, MeasureChar, SetOptions, TextLayout</td>
</tr>
<tr>
<td>for</td>
<td>for</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>foreach</td>
<td>foreach</td>
</tr>
<tr>
<td>format</td>
<td>binary, format, clipboard, selection, Clipboard, CrtSelHdlr, GetSelect</td>
</tr>
<tr>
<td>frame</td>
<td>upvar, frame, ttk_frame, ttk_labelframe</td>
</tr>
<tr>
<td>free</td>
<td>Alloc, DString, Interp, Preserve</td>
</tr>
<tr>
<td>fuzzy comparison</td>
<td>expr</td>
</tr>
<tr>
<td>geometry</td>
<td>wino, wm, GeomReq, GetVRoot, ManageGeom, ttk_Geometry</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td>geometry management</td>
<td>panedwindow</td>
</tr>
<tr>
<td>geometry manager</td>
<td>grid, pack, place, MaintGeom, WindowId</td>
</tr>
<tr>
<td>get</td>
<td>GetSelect</td>
</tr>
<tr>
<td>get variable</td>
<td>SetVar</td>
</tr>
<tr>
<td>glob</td>
<td>glob</td>
</tr>
<tr>
<td>global</td>
<td>global, upvar, variable, Eval</td>
</tr>
<tr>
<td>global variables</td>
<td>SetErrno</td>
</tr>
<tr>
<td>grab</td>
<td>grab, Grab</td>
</tr>
<tr>
<td>graphics context</td>
<td>GetGC</td>
</tr>
<tr>
<td>grid</td>
<td>grid, wm, SetGrid</td>
</tr>
<tr>
<td>group</td>
<td>wm</td>
</tr>
<tr>
<td>groupbox</td>
<td>ttk_labelframe</td>
</tr>
<tr>
<td>grow box</td>
<td>ttk_sizegrip</td>
</tr>
<tr>
<td>handle</td>
<td>event</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>handler</td>
<td>update, selection, Async, CrtChnlHdlr, CrtFileHdlr, CrtTimerHdlr, DoOneEvent, CrtCmHdlr, CrtErrHdlr, CrtGenHdlr, CrtSelHdlr, EventHndlr, HandleEvent, QWinEvent</td>
</tr>
<tr>
<td>hash table</td>
<td>DictObj, Hash</td>
</tr>
<tr>
<td>height</td>
<td>image, place, winfo, ConfigWind, GetVRoot, WindowId</td>
</tr>
<tr>
<td>hidden commands</td>
<td>CrtSlave</td>
</tr>
<tr>
<td>history</td>
<td>history, RecEvalObj, RecordEval</td>
</tr>
<tr>
<td>home directory</td>
<td>Translate</td>
</tr>
<tr>
<td>host</td>
<td>socket</td>
</tr>
<tr>
<td>hostname</td>
<td>GetHostName</td>
</tr>
</tbody>
</table>

Copyright © 1989-1994 The Regents of the University of California
Copyright © 1992-1999 Karl Lehenbauer & Mark Diekhans
Copyright © 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright © 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright © 1994 The Australian National University
Copyright © 1994-2000 Sun Microsystems, Inc
Copyright © 1995-1997 Roger E. Critchlow Jr
Copyright © 1997-2000 Ajuba Solutions
Copyright © 1997-2000 Scriptics Corporation
Copyright © 1998 Mark Harrison
<table>
<thead>
<tr>
<th>i18n</th>
<th>msgcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICCCM</td>
<td>selection</td>
</tr>
<tr>
<td>icon</td>
<td>wm</td>
</tr>
<tr>
<td>iconify</td>
<td>wm</td>
</tr>
<tr>
<td>identifier</td>
<td>winfo, GetHINSTANCE, GetHWND, WindowId</td>
</tr>
<tr>
<td>idle</td>
<td>update, DoOneEvent, Notifier, Inactive</td>
</tr>
<tr>
<td>idle callback</td>
<td>after, DoWhenIdle</td>
</tr>
<tr>
<td>if</td>
<td>if</td>
</tr>
<tr>
<td>illumination</td>
<td>3DBorder</td>
</tr>
<tr>
<td>image</td>
<td>bitmap, image, photo, ttk_image, FindPhoto</td>
</tr>
<tr>
<td>image file</td>
<td>CrtPhImgFmt</td>
</tr>
<tr>
<td>image manager</td>
<td>CrtImgType, DeleteImg, NameOfImg</td>
</tr>
<tr>
<td>image name</td>
<td>NameOfImg</td>
</tr>
<tr>
<td>image size changes</td>
<td>ImgChanged</td>
</tr>
<tr>
<td>image type</td>
<td>CrtImgType</td>
</tr>
<tr>
<td>images</td>
<td>GetImage, ImgChanged</td>
</tr>
<tr>
<td>inactive</td>
<td>Inactive</td>
</tr>
<tr>
<td>inches</td>
<td>GetPixels</td>
</tr>
<tr>
<td>increment</td>
<td>incr</td>
</tr>
<tr>
<td>increments</td>
<td>wm</td>
</tr>
<tr>
<td><strong>index</strong></td>
<td><code>lindex, lrepeat, lset, packagens, pkgMkIndex, string, GetIndex, ListObj</code></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>information</strong></td>
<td><code>info, winfo</code></td>
</tr>
<tr>
<td><strong>initialization</strong></td>
<td><code>AppInit, Init, Tk_Init</code></td>
</tr>
<tr>
<td><strong>initialization procedure</strong></td>
<td><code>StaticPkg</code></td>
</tr>
<tr>
<td><strong>initialized</strong></td>
<td><code>Interp</code></td>
</tr>
<tr>
<td><strong>input</strong></td>
<td><code>chan, OpenFileChnl</code></td>
</tr>
<tr>
<td><strong>insert</strong></td>
<td><code>linsert, ListObj</code></td>
</tr>
<tr>
<td><strong>insertion cursor</strong></td>
<td><code>CanvTxtInfo</code></td>
</tr>
<tr>
<td><strong>instance</strong></td>
<td><code>CrtImgType, GetHINSTANCE</code></td>
</tr>
<tr>
<td><strong>integer</strong></td>
<td><code>ExprLong, ExprLongObj, GetInt, IntObj, LinkVar, ConfigWidg, SetOptions</code></td>
</tr>
<tr>
<td><strong>integer object</strong></td>
<td><code>IntObj</code></td>
</tr>
<tr>
<td><strong>integer type</strong></td>
<td><code>IntObj</code></td>
</tr>
<tr>
<td><strong>intensity</strong></td>
<td><code>GetColor</code></td>
</tr>
<tr>
<td><strong>interactive</strong></td>
<td><code>console</code></td>
</tr>
<tr>
<td><strong>internal</strong></td>
<td><code>namespace</code></td>
</tr>
<tr>
<td><strong>internal representation</strong></td>
<td><code>DoubleObj, IntObj, ListObj, Object, ObjectType, StringObj</code></td>
</tr>
<tr>
<td><strong>internal window</strong></td>
<td><code>CrtWindow</code></td>
</tr>
<tr>
<td><strong>internationalization</strong></td>
<td><code>msgcat, ByteArrObj</code></td>
</tr>
<tr>
<td><strong>interp</strong></td>
<td><code>BackgdErr, SaveResult</code></td>
</tr>
<tr>
<td><strong>interpreter</strong></td>
<td><code>tclsh, info, console, AllowExc, AppInit, AssocData, CallDel, CrtCommand, CrtInterp, CrtSlave, CrtTrace, Init, Interp, Limit, RecEvalObj, RecordEval, SetResult</code></td>
</tr>
<tr>
<td>interpreters</td>
<td>winfo</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
</tr>
<tr>
<td>invoke</td>
<td>CrtSlave</td>
</tr>
<tr>
<td>item type</td>
<td>CanvTkwin, CrtItemType</td>
</tr>
<tr>
<td>iterate</td>
<td>dict</td>
</tr>
<tr>
<td>iteration</td>
<td>continue, for, foreach, DictObj</td>
</tr>
</tbody>
</table>
### Tcl/Tk Keywords - J

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>join</td>
<td>concat, join, SplitPath</td>
</tr>
<tr>
<td>join style</td>
<td>ConfigWidg, GetJoinStl</td>
</tr>
<tr>
<td>justification</td>
<td>GetJustify</td>
</tr>
<tr>
<td>justify</td>
<td>ConfigWidg, SetOptions</td>
</tr>
<tr>
<td>key</td>
<td>AssocData, Hash</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>keyboard</td>
<td>focus</td>
</tr>
<tr>
<td>keyboard events</td>
<td>grab</td>
</tr>
<tr>
<td>keyboard traversal</td>
<td>focusNext</td>
</tr>
<tr>
<td>keysym</td>
<td>keysyms</td>
</tr>
</tbody>
</table>

Copyright © 1989-1994 The Regents of the University of California
Copyright © 1992-1999 Karl Lehenbauer & Mark Diekhans
Copyright © 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright © 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright © 1994 The Australian National University
Copyright © 1994-2000 Sun Microsystems, Inc
Copyright © 1995-1997 Roger E. Critchlow Jr
Copyright © 1997-2000 Ajuba Solutions
Copyright © 1997-2000 Scriptics Corporation
Copyright © 1998 Mark Harrison
Copyright © 2000 Jeffrey Hobbs
Copyright © 2001 ActiveState Tool Corp
Copyright © 2001 Vincent Darley
Copyright © 2001-2004 ActiveState Corporation
Copyright © 2001-2005 Kevin B. Kenny <kennykb(at)acm.org>
Copyright © 2001-2008 Donal K. Fellows
Copyright © 2002-2008 Andreas Kupries <andreas_kupries(at)users.sourceforge.net>
Copyright © 2003 George Petasis <petasis(at)jiit.demokritos.gr>
Copyright © 2003 Simon Geard
Copyright © 2003-2006 Joe English
Copyright © 2006 Miguel Sofer
Copyright © 2006-2007 Daniel A. Steffen <das(at)users.sourceforge.net>
Copyright © 2006-2008 ActiveState Software Inc
Copyright © 2008 Pat Thoyts
<table>
<thead>
<tr>
<th>l10n</th>
<th>msgcat</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>label, ttk_labelframe</td>
</tr>
<tr>
<td>labelframe</td>
<td>labelframe</td>
</tr>
<tr>
<td>length</td>
<td>llength, ListObj</td>
</tr>
<tr>
<td>level</td>
<td>info, uplevel, upvar</td>
</tr>
<tr>
<td>library</td>
<td>library</td>
</tr>
<tr>
<td>limit</td>
<td>Limit</td>
</tr>
<tr>
<td>line</td>
<td>gets</td>
</tr>
<tr>
<td>linemode</td>
<td>fconfigure</td>
</tr>
<tr>
<td>link</td>
<td>LinkVar</td>
</tr>
<tr>
<td>linked variable</td>
<td>UpVar</td>
</tr>
<tr>
<td>list</td>
<td>foreach, join, lappend, lassign, lindex, linsert, list, llength, lrange, lrepeat, lreplace, lreverse, lsearch, lset, lsort, split, ListObj, SetResult, SplitList</td>
</tr>
<tr>
<td>list box</td>
<td>ttk_combobox</td>
</tr>
<tr>
<td>list object</td>
<td>ListObj</td>
</tr>
<tr>
<td>list type</td>
<td>ListObj</td>
</tr>
<tr>
<td>listbox</td>
<td>listbox</td>
</tr>
<tr>
<td>lists</td>
<td>concat</td>
</tr>
<tr>
<td>load</td>
<td>safe, loadTk, Tk_Init</td>
</tr>
<tr>
<td>loading</td>
<td>load</td>
</tr>
<tr>
<td>localization</td>
<td>msgcat</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>location</td>
<td>grid, pack, place, GetVRoot</td>
</tr>
<tr>
<td>lookup</td>
<td>dict, Hash</td>
</tr>
<tr>
<td>loop</td>
<td>break, continue, while</td>
</tr>
<tr>
<td>looping</td>
<td>for, foreach</td>
</tr>
<tr>
<td>lower</td>
<td>lower</td>
</tr>
</tbody>
</table>

Copyright © 1989-1994 The Regents of the University of California
Copyright © 1992-1999 Karl Lehenbauer & Mark Diekhans
Copyright © 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright © 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright © 1994 The Australian National University
Copyright © 1994-2000 Sun Microsystems, Inc
Copyright © 1995-1997 Roger E. Critchlow Jr
Copyright © 1997-2000 Ajuba Solutions
Copyright © 1997-2000 Scriptics Corporation
Copyright © 1998 Mark Harrison
Copyright © 2000 Jeffrey Hobbs
Copyright © 2001 ActiveState Tool Corp
Copyright © 2001 Vincent Darley
Copyright © 2001-2004 ActiveState Corporation
Copyright © 2001-2005 Kevin B. Kenny <kennykb(at)acm.org>
Copyright © 2001-2008 Donal K. Fellows
Copyright © 2002-2008 Andreas Kupries <andreas_kupries(at)users.sourceforge.net>
Copyright © 2003 George Petasis <petasis(at)iit.demokritos.gr>
Copyright © 2003 Simon Geard
Copyright © 2003-2006 Joe English
Copyright © 2006 Miguel Sofer
Copyright © 2006-2007 Daniel A. Steffen <das(at)users.sourceforge.net>
Copyright © 2006-2008 ActiveState Software Inc
Copyright © 2008 Pat Thoyts
<p>| main loop | MainLoop |
| main program | SourceRCFile, Tcl_Main, Tk_Main |
| main window | MainWin, Tk_Init |
| major | GetVersion |
| malloc | Alloc, Interp |
| managed | ManageGeom |
| map | MaintGeom, MapWindow |
| mapped | winfo, WindowId |
| margins | ttk_Geometry |
| master | place, CrtSlave, CrtImgType, MaintGeom |
| master interpreter | interp, safe, loadTk |
| match | lsearch, re_syntax, regexp, regsub, string, switch, RegExp, StrMatch |
| mathematical function | CrtMathFnc |
| measurement | FontId, MeasureChar |
| memory | memory, Alloc, DumpActiveMemory, TCL_MEM_DEBUG |
| menu | menu, popup, ttk_menubutton |
| menubutton | menubutton |
| merge | SplitList |</p>
<table>
<thead>
<tr>
<th>message</th>
<th>msgcat, message</th>
</tr>
</thead>
<tbody>
<tr>
<td>message box</td>
<td>messageBox</td>
</tr>
<tr>
<td>millimeters</td>
<td>ConfigWidg, GetPixels</td>
</tr>
<tr>
<td>minor</td>
<td>GetVersion</td>
</tr>
<tr>
<td>miter</td>
<td>GetJoinStl</td>
</tr>
<tr>
<td>modal</td>
<td>dialog</td>
</tr>
<tr>
<td>modal timeout</td>
<td>QWinEvent</td>
</tr>
<tr>
<td>modules</td>
<td>tm</td>
</tr>
<tr>
<td>Motif compliance</td>
<td>StrictMotif</td>
</tr>
<tr>
<td>move files</td>
<td>file</td>
</tr>
<tr>
<td>multiple</td>
<td>lassign</td>
</tr>
<tr>
<td>mutex</td>
<td>Thread</td>
</tr>
<tr>
<td>name</td>
<td>dde, file, options, send, AddOption, GetOption, GetRelief, Name, SetAppName</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>namespace</td>
<td>global, info, rename, uplevel, upvar, variable, CrtCommand, CrtObjCmd, Namespace</td>
</tr>
<tr>
<td>nesting depth</td>
<td>SetRecLmt</td>
</tr>
<tr>
<td>network address</td>
<td>socket</td>
</tr>
<tr>
<td>newline</td>
<td>fconfigure, puts</td>
</tr>
<tr>
<td>non-blocking</td>
<td>open</td>
</tr>
<tr>
<td>non-existent command</td>
<td>unknown</td>
</tr>
<tr>
<td>nonblocking</td>
<td>close, fblocked, fconfigure, fcopy, fileevent, flush, gets, read, CrtChannel, OpenFileChnl</td>
</tr>
<tr>
<td>nonblocking.</td>
<td>CrtChnlHdlr</td>
</tr>
<tr>
<td>notifier</td>
<td>Notifier</td>
</tr>
<tr>
<td>object</td>
<td>AddErrInfo, BoolObj, ByteArrObj, CrtObjCmd, DictObj, DoubleObj, Eval, ExprLong, ExprLongObj, GetIndex, IntObj, ListObj, Object, ObjectType, RecEvalObj, SetResult, SetVar, StringObj, 3DBorder, BindTable, GetColor</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>object creation</td>
<td>Object</td>
</tr>
<tr>
<td>object result</td>
<td>AddErrInfo</td>
</tr>
<tr>
<td>object type</td>
<td>DoubleObj, IntObj, ListObj, Object, ObjectType, StringObj</td>
</tr>
<tr>
<td>obscure</td>
<td>lower, raise, Restack</td>
</tr>
<tr>
<td>offset</td>
<td>chan</td>
</tr>
<tr>
<td>open</td>
<td>open</td>
</tr>
<tr>
<td>operating system</td>
<td>platform, platform_shell</td>
</tr>
<tr>
<td>operator</td>
<td>mathop</td>
</tr>
<tr>
<td>option</td>
<td>colors, cursors, option, ttk_checkbutton, ttk_radiobutton, ttk_widget, AddOption, GetOption</td>
</tr>
<tr>
<td>option menu</td>
<td>optionMenu</td>
</tr>
<tr>
<td>options</td>
<td>ParseArgv</td>
</tr>
<tr>
<td>order</td>
<td>Isort</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>output</td>
<td>chan, flush, puts, OpenFileChnl</td>
</tr>
<tr>
<td>output channels</td>
<td>console</td>
</tr>
<tr>
<td>own</td>
<td>selection, OwnSelect</td>
</tr>
</tbody>
</table>

Copyright © 1989-1994 The Regents of the University of California
Copyright © 1992-1999 Karl Lehenbauer & Mark Diekhans
Copyright © 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright © 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright © 1994 The Australian National University
Copyright © 1994-2000 Sun Microsystems, Inc
Copyright © 1995-1997 Roger E. Critchlow Jr
Copyright © 1997-2000 Ajuba Solutions
Copyright © 1997-2000 Scriptics Corporation
Copyright © 1998 Mark Harrison
Copyright © 2000 Jeffrey Hobbs
Copyright © 2001 ActiveState Tool Corp
Copyright © 2001 Vincent Darley
Copyright © 2001-2004 ActiveState Corporation
Copyright © 2001-2005 Kevin B. Kenny <kennykb(at)acm.org>
Copyright © 2001-2008 Donal K. Fellow
Copyright © 2002-2008 Andreas Kupries <andreas_kupries(at)users.sourceforge.net>
Copyright © 2003 George Petasis <petasis(at)iit.demokritos.gr>
Copyright © 2003 Simon Geard
Copyright © 2003-2006 Joe English
Copyright © 2006 Miguel Sofer
Copyright © 2006-2007 Daniel A. Steffen <das(at)users.sourceforge.net>
Copyright © 2006-2008 ActiveState Software Inc
Copyright © 2008 Pat Thoyts
<table>
<thead>
<tr>
<th>pack</th>
<th>grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>package</td>
<td>package, packagens, pkgMkIndex, tm, PkgRequire, StaticPkg</td>
</tr>
<tr>
<td>packer</td>
<td>pack</td>
</tr>
<tr>
<td>padding</td>
<td>ttk_Geometry</td>
</tr>
<tr>
<td>palette</td>
<td>palette</td>
</tr>
<tr>
<td>pane</td>
<td>ttk_notebook</td>
</tr>
<tr>
<td>panedwindow</td>
<td>panedwindow</td>
</tr>
<tr>
<td>parcel</td>
<td>pack</td>
</tr>
<tr>
<td>parent</td>
<td>winfo, MaintGeom</td>
</tr>
<tr>
<td>parse</td>
<td>scan, Backslash, ParseCmd, GetScroll</td>
</tr>
<tr>
<td>parsing</td>
<td>regexp</td>
</tr>
<tr>
<td>partial command</td>
<td>CmdCmplt</td>
</tr>
<tr>
<td>patchlevel</td>
<td>GetVersion</td>
</tr>
<tr>
<td>path</td>
<td>SplitPath, CanvPsY</td>
</tr>
<tr>
<td>path name</td>
<td>winfo, Name</td>
</tr>
<tr>
<td>pattern</td>
<td>glob, lsearch, regexp, regsub, string, RegExp, StrMatch</td>
</tr>
<tr>
<td>permissions</td>
<td>open, GetOpnFl</td>
</tr>
<tr>
<td>photo</td>
<td>photo, FindPhoto</td>
</tr>
<tr>
<td>photo image</td>
<td>CrtPhImgFmt</td>
</tr>
<tr>
<td>pipeline</td>
<td>exec, open, pid, GetOpnFl</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>pixel</td>
<td>ConfigWind</td>
</tr>
<tr>
<td>pixel value</td>
<td>GetColor</td>
</tr>
<tr>
<td>pixels</td>
<td>ConfigWidg, GetPixels, SetOptions</td>
</tr>
<tr>
<td>pixmap</td>
<td>ConfigWidg, GetBitmap, GetPixmap</td>
</tr>
<tr>
<td>pixmap theme</td>
<td>ttk_image</td>
</tr>
<tr>
<td>place</td>
<td>place</td>
</tr>
<tr>
<td>platform</td>
<td>fconfigure, platform, platform_shell</td>
</tr>
<tr>
<td>platform-specific</td>
<td>chooseDirectory</td>
</tr>
<tr>
<td>pointer events</td>
<td>grab</td>
</tr>
<tr>
<td>points</td>
<td>GetPixels</td>
</tr>
<tr>
<td>polygon</td>
<td>3DBorder</td>
</tr>
<tr>
<td>popup</td>
<td>popup</td>
</tr>
<tr>
<td>portability</td>
<td>filename</td>
</tr>
<tr>
<td>position</td>
<td>wm, MaintGeom, MoveToplev</td>
</tr>
<tr>
<td>POSIX</td>
<td>tclvars</td>
</tr>
<tr>
<td>Postscript</td>
<td>CanvPsY, FontId</td>
</tr>
<tr>
<td>precision</td>
<td>tclvars</td>
</tr>
<tr>
<td>present</td>
<td>PkgRequire</td>
</tr>
<tr>
<td>priority</td>
<td>option</td>
</tr>
<tr>
<td>procedure</td>
<td>apply, global, info, proc, return, upvar, variable</td>
</tr>
<tr>
<td>process</td>
<td>exit, open, DetachPids</td>
</tr>
<tr>
<td>process identifier</td>
<td>pid</td>
</tr>
<tr>
<td>projecting</td>
<td>GetCapStyl</td>
</tr>
<tr>
<td>prompt</td>
<td>tclsh</td>
</tr>
<tr>
<td>propagation</td>
<td>grid, pack</td>
</tr>
</tbody>
</table>
provide  PkgRequire
pwd  GetCwd
<table>
<thead>
<tr>
<th>radiobutton</th>
<th>radiobutton</th>
</tr>
</thead>
<tbody>
<tr>
<td>raise</td>
<td>raise</td>
</tr>
<tr>
<td>raised</td>
<td>3DBorder</td>
</tr>
<tr>
<td>range</td>
<td>lrange</td>
</tr>
<tr>
<td>rc file</td>
<td>SourceRCFile</td>
</tr>
<tr>
<td>read</td>
<td>fcopy, gets, read, set, trace, GetOpenFl, OpenFileChnl</td>
</tr>
<tr>
<td>read-only</td>
<td>LinkVar</td>
</tr>
<tr>
<td>readable</td>
<td>fileevent</td>
</tr>
<tr>
<td>real</td>
<td>LinkVar</td>
</tr>
<tr>
<td>realloc</td>
<td>Alloc</td>
</tr>
<tr>
<td>record</td>
<td>history, RecEvalObj, RecordEval</td>
</tr>
<tr>
<td>recursion</td>
<td>SetRecLmt</td>
</tr>
<tr>
<td>redirection</td>
<td>exec</td>
</tr>
<tr>
<td>redisplay</td>
<td>CanvTkwin, GetImage, ImgChanged</td>
</tr>
<tr>
<td>reference count</td>
<td>Preserve</td>
</tr>
<tr>
<td>reference counting</td>
<td>Object</td>
</tr>
<tr>
<td>reflection</td>
<td>refchan</td>
</tr>
<tr>
<td>region</td>
<td>ttk_Geometry</td>
</tr>
<tr>
<td>register</td>
<td>SetAppName</td>
</tr>
<tr>
<td>registry</td>
<td>registry</td>
</tr>
<tr>
<td>regular expression</td>
<td>lsearch, re_syntax, \nregexp, regsub, switch, RegExp</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------</td>
</tr>
<tr>
<td>relative file name</td>
<td>filename</td>
</tr>
<tr>
<td>release</td>
<td>GetVersion</td>
</tr>
<tr>
<td>relief</td>
<td>ConfigWidg, GetRelief, SetOptions, ttk_Geometry</td>
</tr>
<tr>
<td>remote execution</td>
<td>dde, send</td>
</tr>
<tr>
<td>remove</td>
<td>unset</td>
</tr>
<tr>
<td>rename</td>
<td>rename, trace</td>
</tr>
<tr>
<td>rename files</td>
<td>file</td>
</tr>
<tr>
<td>replace</td>
<td>lreplace, lset, ListObj</td>
</tr>
<tr>
<td>reporting</td>
<td>bgerror, tkerror</td>
</tr>
<tr>
<td>request</td>
<td>GeomReq, ManageGeom</td>
</tr>
<tr>
<td>requested size</td>
<td>WindowId</td>
</tr>
<tr>
<td>require</td>
<td>PkgRequire</td>
</tr>
<tr>
<td>resource</td>
<td>Limit</td>
</tr>
<tr>
<td>resource identifier</td>
<td>FreeXId, GetPixmap</td>
</tr>
<tr>
<td>restriction</td>
<td>RestrictEv</td>
</tr>
<tr>
<td>result</td>
<td>DString, Eval, Interp, SaveResult, SetResult</td>
</tr>
<tr>
<td>retrieve</td>
<td>option, GetOption</td>
</tr>
<tr>
<td>return</td>
<td>return</td>
</tr>
<tr>
<td>return value</td>
<td>SetResult</td>
</tr>
<tr>
<td>reverse</td>
<td>lreverse, string</td>
</tr>
<tr>
<td>ring</td>
<td>bell</td>
</tr>
<tr>
<td>root window</td>
<td>CoordToWin, GetRootCrd</td>
</tr>
<tr>
<td>round</td>
<td>GetCapStyl, GetJoinStl</td>
</tr>
<tr>
<td>rubber sheet</td>
<td>place</td>
</tr>
<tr>
<td>safe</td>
<td>Tk_Init</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>safe interpreter</td>
<td>safe, loadTk</td>
</tr>
<tr>
<td>safe interpreter</td>
<td>interp, load, unload</td>
</tr>
<tr>
<td>scalar</td>
<td>SetVar</td>
</tr>
<tr>
<td>scale</td>
<td>scale, ttk_scale</td>
</tr>
<tr>
<td>scan</td>
<td>binary, scan</td>
</tr>
<tr>
<td>screen</td>
<td>wininfo, CrtWindow, GetVisual, WindowId</td>
</tr>
<tr>
<td>screen distance</td>
<td>SetOptions</td>
</tr>
<tr>
<td>screen units</td>
<td>GetPixels</td>
</tr>
<tr>
<td>script</td>
<td>eval, fileevent, source, time, Eval, BindTable</td>
</tr>
<tr>
<td>script file</td>
<td>tclsh</td>
</tr>
<tr>
<td>scrollbar</td>
<td>scrollbar, ttk_scrollbar, GetScroll</td>
</tr>
<tr>
<td>scrolling command</td>
<td>GetScroll</td>
</tr>
<tr>
<td>search</td>
<td>array, Isearch, Hash</td>
</tr>
<tr>
<td>security</td>
<td>send</td>
</tr>
<tr>
<td>security policy</td>
<td>http</td>
</tr>
<tr>
<td>seek</td>
<td>seek, OpenFileChnl</td>
</tr>
<tr>
<td>seeking</td>
<td>tell</td>
</tr>
<tr>
<td>selection</td>
<td>chooseDirectory, clipboard, selection, CanvTkwin, CanvTxtInfo,</td>
</tr>
<tr>
<td>Term</td>
<td>Code/Command</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>selection anchor</td>
<td>CanvTxtInfo</td>
</tr>
<tr>
<td>selection owner</td>
<td>OwnSelect</td>
</tr>
<tr>
<td>selection retrieval</td>
<td>GetSelect</td>
</tr>
<tr>
<td>send</td>
<td>send, tk</td>
</tr>
<tr>
<td>send command</td>
<td>SetAppName</td>
</tr>
<tr>
<td>separator</td>
<td>join, ttk_separator</td>
</tr>
<tr>
<td>serial</td>
<td>open</td>
</tr>
<tr>
<td>server</td>
<td>OpenTcp</td>
</tr>
<tr>
<td>service mode</td>
<td>Notifier</td>
</tr>
<tr>
<td>set</td>
<td>lassign, lset, SetVar</td>
</tr>
<tr>
<td>shadow</td>
<td>3DBorder</td>
</tr>
<tr>
<td>shared library</td>
<td>load, unload</td>
</tr>
<tr>
<td>shell</td>
<td>tclsh, wish</td>
</tr>
<tr>
<td>signal</td>
<td>Async</td>
</tr>
<tr>
<td>signal numbers</td>
<td>Signal</td>
</tr>
<tr>
<td>signals</td>
<td>Signal</td>
</tr>
<tr>
<td>size</td>
<td>grid, pack, wm</td>
</tr>
<tr>
<td>sizegrip</td>
<td>ttk_sizegrip</td>
</tr>
<tr>
<td>slave</td>
<td>place, CrtSlave, MaintGeom</td>
</tr>
<tr>
<td>slave interpreter</td>
<td>interp, safe, loadTk</td>
</tr>
<tr>
<td>sleep</td>
<td>after, Sleep</td>
</tr>
<tr>
<td>slider</td>
<td>scale, ttk_scale</td>
</tr>
<tr>
<td>socket</td>
<td>http, socket</td>
</tr>
<tr>
<td>sort</td>
<td>lsort</td>
</tr>
<tr>
<td>source</td>
<td>safe, loadTk</td>
</tr>
<tr>
<td>spinbox</td>
<td>spinbox</td>
</tr>
<tr>
<td>split</td>
<td>split, SplitList, SplitPath</td>
</tr>
<tr>
<td>splitting</td>
<td>regexp</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>sprintf</td>
<td>format</td>
</tr>
<tr>
<td>stack</td>
<td>AddErrInfo</td>
</tr>
<tr>
<td>stack frame</td>
<td>uplevel</td>
</tr>
<tr>
<td>stacking order</td>
<td>lower, raise, Restack</td>
</tr>
<tr>
<td>standard channel</td>
<td>GetStdChan</td>
</tr>
<tr>
<td>standard channels</td>
<td>StdChannels, CrtConsoleChan</td>
</tr>
<tr>
<td>standard error</td>
<td>GetStdChan</td>
</tr>
<tr>
<td>standard input</td>
<td>GetStdChan</td>
</tr>
<tr>
<td>standard option</td>
<td>options</td>
</tr>
<tr>
<td>standard output</td>
<td>GetStdChan</td>
</tr>
<tr>
<td>stat</td>
<td>file, Access, FileSystem</td>
</tr>
<tr>
<td>state</td>
<td>ttk_widget, SaveResult</td>
</tr>
<tr>
<td>static linking</td>
<td>StaticPkg</td>
</tr>
<tr>
<td>sticky</td>
<td>ttk_Geometry</td>
</tr>
<tr>
<td>stipple</td>
<td>CanvPsY</td>
</tr>
<tr>
<td>storage</td>
<td>Preserve</td>
</tr>
<tr>
<td>string</td>
<td>format, lsearch, re_syntax, regexp, split, string, ExprLong, ExprLongObj, LinkVar, PrintDbf, RegExp, StrMatch, GetJustify, GetRelief</td>
</tr>
<tr>
<td>string object</td>
<td>StringObj</td>
</tr>
<tr>
<td>string representation</td>
<td>DoubleObj, IntObj, ListObj, Object, ObjectType, StringObj</td>
</tr>
<tr>
<td>string type</td>
<td>StringObj</td>
</tr>
<tr>
<td>strings</td>
<td>Concat, SplitList</td>
</tr>
<tr>
<td>stubs</td>
<td>InitStubs, TkInitStubs</td>
</tr>
<tr>
<td>style</td>
<td>ttk_image, ttk_style, ttk_vsapi</td>
</tr>
<tr>
<td>subexpression</td>
<td>RegExp</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>sublist</td>
<td>rrange</td>
</tr>
<tr>
<td>subprocess</td>
<td>exec, tclvars</td>
</tr>
<tr>
<td>substitute</td>
<td>regsub</td>
</tr>
<tr>
<td>substitution</td>
<td>format</td>
</tr>
<tr>
<td>switch</td>
<td>switch, options</td>
</tr>
<tr>
<td>synonym</td>
<td>ConfigWidg, SetOptions</td>
</tr>
</tbody>
</table>

Copyright © 1989-1994 The Regents of the University of California
Copyright © 1992-1999 Karl Lehenbauer & Mark Diekhans
Copyright © 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright © 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright © 1994 The Australian National University
Copyright © 1994-2000 Sun Microsystems, Inc
Copyright © 1995-1997 Roger E. Critchlow Jr
Copyright © 1997-2000 Ajuba Solutions
Copyright © 1997-2000 Scriptics Corporation
Copyright © 1998 Mark Harrison
Copyright © 2000 Jeffrey Hobbs
Copyright © 2001 ActiveState Tool Corp
Copyright © 2001 Vincent Darley
Copyright © 2001-2004 ActiveState Corporation
Copyright © 2001-2005 Kevin B. Kenny <kennykb(at)acm.org>
Copyright © 2001-2008 Donal K. Fellows
Copyright © 2002-2008 Andreas Kupries <andreas_kupries(at)users.sourceforge.net>
Copyright © 2003 George Petasis <petasis(at)iit.demokritos.gr>
Copyright © 2003 Simon Geard
Copyright © 2003-2006 Joe English
Copyright © 2006 Miguel Sofer
Copyright © 2006-2007 Daniel A. Steffen <das(at)users.sourceforge.net>
Copyright © 2006-2008 ActiveState Software Inc
Copyright © 2008 Pat Thoyts
<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab</td>
<td>GetIndex</td>
</tr>
<tr>
<td>table lookup</td>
<td>alloc</td>
</tr>
<tr>
<td>tag</td>
<td>bindtags</td>
</tr>
<tr>
<td>target</td>
<td>selection, CrtSelHdr</td>
</tr>
<tr>
<td>TCL_MEM_DEBUG</td>
<td>Alloc</td>
</tr>
<tr>
<td>Tcl_RegexIndices</td>
<td>RegExp</td>
</tr>
<tr>
<td>Tcl_RegexInfo</td>
<td>RegExp</td>
</tr>
<tr>
<td>Tcl_SaveInterpState</td>
<td>Async</td>
</tr>
<tr>
<td>TCP</td>
<td>OpenTcp</td>
</tr>
<tr>
<td>tcp</td>
<td>socket</td>
</tr>
<tr>
<td>test</td>
<td>tcltest, while</td>
</tr>
<tr>
<td>test harness</td>
<td>tcltest</td>
</tr>
<tr>
<td>test suite</td>
<td>tcltest</td>
</tr>
<tr>
<td>text</td>
<td>msgcat, text, tkvars, CanvTxtInfo</td>
</tr>
<tr>
<td>text box</td>
<td>ttk_combobox</td>
</tr>
<tr>
<td>text field</td>
<td>ttk_entry</td>
</tr>
<tr>
<td>theme</td>
<td>ttk_image, ttk_style, ttk_vsapi</td>
</tr>
<tr>
<td>thread</td>
<td>Async, Exit, Thread</td>
</tr>
<tr>
<td>thread local storage</td>
<td>Thread</td>
</tr>
<tr>
<td>threads</td>
<td>Notifier</td>
</tr>
<tr>
<td>three-dimensional effect</td>
<td>3DBorder</td>
</tr>
<tr>
<td>tilde</td>
<td>Translate</td>
</tr>
<tr>
<td>time</td>
<td>after, clock, time, GetTime, Limit, Sleep</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>timer</td>
<td>CrtTimerHdlr, DoOneEvent, Notifier</td>
</tr>
<tr>
<td>title</td>
<td>wm</td>
</tr>
<tr>
<td>tk_strictMotif variable</td>
<td>StrictMotif</td>
</tr>
<tr>
<td>tkvars</td>
<td>text</td>
</tr>
<tr>
<td>toggle</td>
<td>ttk_checkbutton</td>
</tr>
<tr>
<td>token</td>
<td>ParseCmd, Name</td>
</tr>
<tr>
<td>tolower</td>
<td>ToUpper</td>
</tr>
<tr>
<td>toolkit</td>
<td>wish</td>
</tr>
<tr>
<td>top-level</td>
<td>focus, focusNext, WindowId</td>
</tr>
<tr>
<td>top-level window</td>
<td>wm, CrtWindow, MoveToplev</td>
</tr>
<tr>
<td>toplevel</td>
<td>toplevel</td>
</tr>
<tr>
<td>totitle</td>
<td>ToUpper</td>
</tr>
<tr>
<td>toupper</td>
<td>ToUpper</td>
</tr>
<tr>
<td>trace</td>
<td>trace, AddErrInfo, CrtTrace, TraceCmd, TraceVar</td>
</tr>
<tr>
<td>traces</td>
<td>LinkVar</td>
</tr>
<tr>
<td>translate</td>
<td>Translate</td>
</tr>
<tr>
<td>translation</td>
<td>fconfigure, fcopy, msgcat, read</td>
</tr>
<tr>
<td>traversal highlight</td>
<td>DrawFocHlt</td>
</tr>
<tr>
<td>trough</td>
<td>scale, ttk_scale</td>
</tr>
<tr>
<td>true</td>
<td>if</td>
</tr>
<tr>
<td>type</td>
<td>clipboard, selection, SplitPath, Clipboard</td>
</tr>
<tr>
<td>type conversion</td>
<td>Object, ObjectType</td>
</tr>
<tr>
<td>type manager</td>
<td>CanvTkwin, CrtItemType</td>
</tr>
</tbody>
</table>
types of images
<table>
<thead>
<tr>
<th>uid</th>
<th>ConfigWidg</th>
</tr>
</thead>
<tbody>
<tr>
<td>unicode</td>
<td>ByteArrObj, StringObj, ToUpper, UniCharIsAlpha, Utf</td>
</tr>
<tr>
<td>unique identifier</td>
<td>GetUid, SetClass</td>
</tr>
<tr>
<td>units</td>
<td>wm</td>
</tr>
<tr>
<td>unknown</td>
<td>library</td>
</tr>
<tr>
<td>unloading</td>
<td>unload, Exit</td>
</tr>
<tr>
<td>unmanaged</td>
<td>ManageGeom</td>
</tr>
<tr>
<td>unmap</td>
<td>MaintGeom, MapWindow</td>
</tr>
<tr>
<td>unset</td>
<td>trace, SetVar</td>
</tr>
<tr>
<td>update</td>
<td>dict, update</td>
</tr>
<tr>
<td>upvar</td>
<td>UpVar</td>
</tr>
<tr>
<td>user</td>
<td>Translate</td>
</tr>
<tr>
<td>utf</td>
<td>ByteArrObj, Encoding, ToUpper, Utf</td>
</tr>
</tbody>
</table>

Copyright © 1989-1994 The Regents of the University of California
Copyright © 1992-1999 Karl Lehenbauer & Mark Diekhans
Copyright © 1992-1999 Karl Lehenbauer and Mark Diekhans
Copyright © 1993-1997 Bell Labs Innovations for Lucent Technologies
Copyright © 1994 The Australian National University
Copyright © 1994-2000 Sun Microsystems, Inc
Copyright © 1995-1997 Roger E. Critchlow Jr
Copyright © 1997-2000 Ajuba Solutions
Copyright © 1997-2000 Scriptics Corporation
Copyright © 1998 Mark Harrison
Copyright © 2000 Jeffrey Hobbs
<table>
<thead>
<tr>
<th>value</th>
<th>incr, Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable</td>
<td>append, global, incr, info, lappend, lassign, namespace, set, trace, unset, upvar, variable, vwait, tkwait, AddErrInfo, Environment, LinkVar, SetVar, TraceVar, UpVar</td>
</tr>
<tr>
<td>variable substitution</td>
<td>subst, ParseCmd, SubstObj</td>
</tr>
<tr>
<td>variables</td>
<td>tclvars, uplevel, tkvars</td>
</tr>
<tr>
<td>version</td>
<td>package, packagens, pkgMkIndex, tkvars, GetVersion, PkgRequire</td>
</tr>
<tr>
<td>vfs</td>
<td>FileSystem</td>
</tr>
<tr>
<td>virtual</td>
<td>FileSystem</td>
</tr>
<tr>
<td>virtual event</td>
<td>event</td>
</tr>
<tr>
<td>virtual root</td>
<td>winfo, GetVRoot</td>
</tr>
<tr>
<td>visibility</td>
<td>tkwait</td>
</tr>
<tr>
<td>visual</td>
<td>GetClrmap, GetVisual, SetVisual, WindowId</td>
</tr>
<tr>
<td>volume-relative file name</td>
<td>filename</td>
</tr>
</tbody>
</table>
### Tcl/Tk Keywords - W

<table>
<thead>
<tr>
<th>Wait</th>
<th>vwait, tkwait, DetachPids, Sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>While</td>
<td>while</td>
</tr>
<tr>
<td>Whitespace</td>
<td>library</td>
</tr>
<tr>
<td>Widget</td>
<td>button, canvas, checkbutton, entry, frame, label, labelframe, listbox, menu, menubutton, message, panedwindow, radiobutton, scale, scrollbar, spinbox, text, toplevel, ttk_button, ttk_checkbutton, ttk_combobox, ttk_entry, ttk_frame, ttk_labelframe, ttk_menubutton, ttk_radiobutton, ttk_scale, ttk_scrollbar, ttk_separator, ttk_sizegrip</td>
</tr>
<tr>
<td>Width</td>
<td>image, place, winfo, ConfigWind, GetVRoot, WindowId</td>
</tr>
<tr>
<td>Window</td>
<td>console, destroy, grab, tkwait, winfo, ConfigWind, CrtWindow, GetHWND, Grab, HandleEvent, MapWindow, Name,</td>
</tr>
<tr>
<td>window manager</td>
<td>SetClass, SetGrid, WindowId</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>windows</td>
<td>ttk_vsapi</td>
</tr>
<tr>
<td>Windows window id</td>
<td>HWNDToWindow</td>
</tr>
<tr>
<td>word</td>
<td>library, string</td>
</tr>
<tr>
<td>writable.</td>
<td>fileevent</td>
</tr>
<tr>
<td>write</td>
<td>puts, set, trace, GetOpnFl, OpenFileChnl</td>
</tr>
</tbody>
</table>

**wrong number of arguments**

WrongNumArgs
<table>
<thead>
<tr>
<th>x</th>
<th>ConfigWind, WindowId</th>
</tr>
</thead>
<tbody>
<tr>
<td>X window id</td>
<td>IdToWindow</td>
</tr>
<tr>
<td>xview</td>
<td>GetScroll</td>
</tr>
</tbody>
</table>
y

yview

ConfigWind, WindowId

GetScroll