RX Library Help v1.1.10 (both on-line and off-line edition)

RxLibrary Set of Native Delphi Components for Borland Delphi versions 1, 2, 3, 4 and 5 and Borland C++ Builder 1.0, 2.0, 4.0 and 5.0 with **full** Source Code.

The library and help documentation is separate and have different version numbering. Current RxLibrary Version: 2.75 Current Help Version: 1.1.10

The off-line edition is now available. Do check the <u>download</u> site every month to ensure you always have the latest version of RxLibrary and it's English Help Documentation.

Notes:

<u>Installation Notes...</u> <u>Tips with using RxLibrary</u> <u>Common Problems with Installing RxLibrary</u>

Sort by Component Library, Filenames or by Classes...

Class - Components	Brief Description	File
<u>TRxComboEdit</u>	Combo Edit field with a button.	<u>ToolEdit</u>
<u>TRxFilenameEdit</u>	Edit Field with button to select a filename	<u>ToolEdit</u>
<u>TRxDateEdit</u>	Edit Field designed with date input and calendar	<u>ToolEdit</u>
TRxDirectoryEdit	Edit field with a pop-up directory selection dialogue box	<u>ToolEdit</u>
<u>TRxCurrencyEdit</u>	Edit field with currency editing properties	<u>CurrEdit</u>
<u>TRxTextListBox</u>	TListBox descendant with optional horizontal scrollbar.	<u>RxCtrls</u>
<u>TRxCheckListBox</u>	Scrollable TListBox with check boxes next to each item.	<u>RXCtrls</u>
TRxFontComboBox	TListBox displaying Fonts.	<u>RxCombos</u>
TRxColorComboBox	TListBox displaying a Colour combination.	RxCombos
<u>TRxSplitter</u>	Splitter component.	<u>RxSplit</u>
<u>TRxSlider</u>	Slider Component	<u>RxSlider</u>
<u>TRxLabel</u>	TLabel with 3D special effects and other options.	<u>RxCtrls</u>
<u>TRxClock</u>	Standard digit and analogue Clock	<u>RxClock</u>
TRxDrawGrid	Draw grid	<u>RxCtrls</u>
TRxSpeedButton	Speed button with an explorer style alternative.	<u>RxSpeed</u>
<u>TRxSpinEdit</u>	Spinner button	<u>RxSpin</u>
<u>TRxSwitch</u>	The alternative to the Delphi 1.0 VBX switch	<u>RxSwitch</u>
<u>TRxDice</u>	Dice	<u>RxDice</u>
Class - Database	Brief Description	File
TRxQuery, TSQLScript	Special version of Query and SQL Execute	<u>RxQuery</u>
TMemory Table	In-Line memory table to store temporary queries	<u>MemTable</u>
TQBEQuery	Query by Example (see your academic notes)	<u>DbQbe</u>
TRxDBFilter	Database filter	<u>DbFilter</u>

TRxDBGrid	Data-aware <u>RxGrid</u> .	<u>DbGrid</u>
<u>TDBDateEdit, PickDate</u>	Data Aware <u>RX DateEdit</u>	<u>DBCtrl</u>
TDBStatusLabel	Information about the status of an open database file.	RxDBComb
TRxDBLookupList	Similar to InfoPower's Woll2Woll components.	RxDBComb
TRxDBLookupCombo	Similar to InfoPower's Woll2Woll components.	<u>RxLookup</u>
TRxDBComboBox	Similar to InfoPower's Woll2Woll components.	RxDBComb
TRxDBRichEdit	Data-Aware RichEdit control	DBRichEd
TDBIndexCombo	Provides a drop-down list of the indexes.	DBIndex
TDBProgress	Monitor a "remote" (or process on another computer) SQL procedure	DBPrgrss
TDBSecurity	Database Security options	DBSecur
TRxBdeErrorDlg	Replaces the standard BDE data-aware Error message	<u>DBExcpt</u>
Class	Brief Description	File
<u>TPicClip</u>	Provides the functionality of the PICCLIP.VBX control which ships with Visual Basic Professional	<u>PicClip</u>
TFormPlacement	The TFormPlacement component provides a quick method for saving size and position of its parent form.	<u>Placement</u>
TFormStorage	Stores the form's internal details	<u>Placement</u>
<u>TRxIniFile</u>	The lower level details of <u>RX Form Storage</u> .	<u>RxIni</u>
TRxWindowHook	Make an extension to a component that does not have the source code, or is difficult to modify.	<u>RxHook</u>
TDualListDialog	Allows creating a dual-list box (e.g., installation settings)	<u>DualList</u>
<u>TConverter</u>	Multiple purpose converter VCL wrapper	DataConv
TSecretPanel	Similar to tPanel, except that it can be used as a scroller	<u>RxCtrls</u>
<u>TSpeedbar</u>	A dockable speedbar	<u>Speedbar</u>
<u>TClipboardViewer</u>	Display the contents of the Windows Clipboard on a form	<u>ClipView</u>
<u>TPageManager</u>	Create wizards	PageMngr
TRxCalculator	Calculator (decimal and floating point edition)	<u>RxCalc</u>
TRxStrHolder	A VCL container for stringlist.	<u>StrHolder</u>
<u>TAppEvent</u>	General Purpose Application handler	<u>AppEvent</u>
<u>TRxTimerList</u>	Timer "list of list" (to conserve timer resources)	<u>TimerLst</u>
<u>TMRUManager</u>	Most Recently Used Manager (e.g., Browser back and forward buttons?)	<u>MRUList</u>
<u>TRxTrayIcon</u>	Tray Icon - and note the ShellAPI extensions it contains (This is used in combination with <u>AppEvent</u>)	<u>RxShell</u>
<u>TIconList</u>	Internal details and workings of the <u>RX Tray Icon</u> .	<u>IcoList</u>
TAnimatedCursorImage	Animated Cursors	<u>AniFile</u>
TAnimateImage	Animation	<u>Animate</u>
<u>TRxMathParser</u>	Mathematical parser	Parsing
<u>TVerInfo</u>	Version Information Stuff	<u>VerInfo</u>

Description	Brief Description	File
Application Units	Visual Application Events (no coding instead)	<u>AppUtils</u>
TDBLocate	Borland Database Engine Utilities	<u>BdeUtils</u>
<u>TLocateObject</u>	Database Utilities	DBUtils
Box Procedures	Box Procedures	BoxProcs
<u>Date Util</u>	Date Utilities	<u>DateUtil</u>
File Utilities	File Utilities	<u>FileUtil</u>
<u>Min - Max</u>	Minimum and Maximum.	<u>MaxMin</u>
<u>THugeList</u>	Huge Object Oriented List	<u>ObjStr</u>
RX Ole 2 Automation	OLE2 Automation stuff.	<u>Ole2Auto</u>
RX StrUtils	String Utilities	StrUtils
<u>RX Splash Windows</u>	Splash window	<u>SplshWnd</u>
RX VCL Utilities	VCL Utilities	<u>VCLUtils</u>
RX Clip Icons	Code for the clipboard.	<u>ClipIcon</u>
Class	Brief Description	File
TRxGIFAnimator	GIF Animation Routine	<u>GIFCtrl</u>
TGIFFrame	GIF routines	<u>RxGIF</u>
TRxDBCalcEdit	Data-aware Edit field with <u>RxCalculator</u>	<u>RxDBCtrl</u>
<u>TRxCalcEdit</u>	Edit field with <u>RxCalculator</u>	RxCombos
<u>RxGraph</u>	Bitmap to memory formatting	<u>RX Graph</u>

Download

Please download the latest version of Rx Library Help (off-line edition) at:

http://helpmaker.hypermart.net/rxlib/rxlib.zip

Installation Notes for Installing RxHelp version 2.4

Run RXINST.EXE.

Before installing RX Library components into Delphi, check out RX.INC file located in RX\UNITS subdirectory. This file is included in all RX Library units and contains conditional defines that affects compilation. You can change some of these defines or specify global compiler options there.

Delphi 3.x:

Remove previously compiled RX packages RXCTL.DPL, RXDB.DPL, RXTOOLS.DPL, DCLRXCTL.DPL, DCLRXDB.DPL and DCLRXTLS.DPL from your hard disk.

Use "File\Open..." menu item of Delphi IDE to open consistently RX run-time packages RXCTL.DPK (MUST be first), RXDB.DPK and RXTOOLS.DPK. In "Package.." window click "Compile" button to compile RX Library run-time packages. Put compiled DPL files into directory that is accessible through the search PATH (for example, in the Windows\System directory). After compiling RX run-time packages you must install RX design-time packages into the IDE.

Use "File\Open..." menu item to open consistently RX design-time packages DCLRXCTL.DPK (MUST be first), DCLRXDB.DPK and DCLRXTLS.DPK. In "Package.." window click "Install" button to register RX Library components on the "RX Controls", "RX DBAware" and "RX Tools" pages accordingly.

Delphi 2.x and C++ Builder 1.0:

Use the "Install..." item on Delphi's "Component" menu to add the RxCtlReg.PAS, RxDBReg.PAS and RxTooReg.PAS units to the component library. These units registers all RX Library components on the "RX Controls", "RX DBAware" and "RX Tools" pages accordingly.

Delphi 1.x:

Use the "Install Components..." item on Delphi's Options menu to add the RxCtlReg.PAS, RxDBReg.PAS and RxTooReg.PAS units to the component library. These units registers all RX Library components on the "RX Controls", "RX DBAware" and "RX Tools" pages accordingly.

Last of all, ensure that your RxLibrary path is inside the Delphi environment path. If you don't succeed or there's unusual problem to add

Loading from Resources.

I'm mixed about using the *resources* section of the EXE file to load GIFs.

Against:

The reason is that you can be sure that nobody will be stealing those images $\langle g \rangle$ and do not have to worry about distributing the *correct* resource version. Instead of using resources, you may want to use <u>Rx Picture</u> <u>Clip</u> instead.

For:

You would like to store the information as a resource DLL.

Using Rx Picture Clip instead of Delphi's default Image List.

The <u>Rx Picture Clip</u> has a definite advantage over Delphi's Win32 Image List.

For starters, it does not use the ComCtrl32.DLL which causes your images to disappear when the ComCtrl32.DLL is the wrong build on your client's computer. As the images are self contained inside the EXE file, instead of the resources, you can be sure that nobody will be stealing those images <g>.

By using the <u>Rx Picture Clip</u>, you do not have to worry about distributing the *correct* version ComCtrl32.DLL

Explorer Button Substitution.

The **<u>RxSpeedbutton</u>** has an interesting property known as <u>**Flat**</u> which allows it to mimic MSIE's explorer button. Further to this, you can add the speed buttons to the <u>**RxSpeedBar**</u>.

Tray Button and Form Hiding

The <u>TrayIcon</u> property puts the image on Windows 95 (or NT's) "tray". However, nothing is mentioned about hiding the form.

Drop an <u>AppEvents</u> component to your form (make sure this is your 1st or main form) and set the <u>ShowMainForm</u> property to false to hide it.

Remember to add a *popup menu* or a *hints description* to TrayIcon. The popup menu would access another form that would be the applications form.

In other words,

- 1. Place an <u>AppEvents</u> component to your main form.
- 2. Place a <u>TrayIcon</u> component to your form.
- 3. Set the AppEvent component <u>ShowMainForm</u> property to false.
- 4. Add a popup menu to TrayIcon.Popupmenu property.
- 5. The popup menu should show another form. This form is the main application.

Tips

Saving the Form's Location...

Use the <u>RX Form Placement</u> component. Specify what properties you want to save in the <u>Options property</u>. For this to be done automatically, set <u>Active</u> to True.

Common Installation Problems and Known Bugs

Q: Each time I try to recompile the library I get an error saying that "SHLOBJ" is missing while processing the FileUtil.pas file. I have tried searching throughout my system for ANY reference of this file. Is there a update that I missed?

A: SHLOBJ is a file that is part of Delphi 2.0 OpenApi. Hence, it is not distributed with Delphi. It is only distributed as a DCU file. You will have to add the path where SHLOBJ.DCU is into your environment path to compile Rx Library.

Addendum: Delphi 2.0 definitely does not have such a unit. To install RXLibrary 2.40 into Delphi 2.0 download patched version of FILEUTIL.PAS unit: **FileUtil.pas unit for Delphi 2.0** [futil240.zip]

Q: I've just installed the RxLib but got some error running their DEMO. The error related to the component TDateEdit. When I run the DEMO, I got:-

..\ToolEdit.pas (1838): Undeclared Identifier: 'CreatePopupCalender' ..\ToolEdit.pas (1995): Undeclared Identifier: 'SetupPopupCalender'

A: One of RxLibrary's filename for the date operation is called **PickDate.pas**, which is a modified version of a version found in \Delphi\Demo\Db\MastApp\PickDate.pas. Check or examine your Delphi environment paths for any duplicate 'PickDate.pas' filenames. If found, please rename these duplicates so they will not conflict with Rx library version of PickDate.Pas

Q: Using the TRxDBComboEdit component from RX 2.40 gave some trouble when hitting the button. The event OnButtonClick works only when the dataset is in edit-mode. **A:** You need to rewrite TRxDBComboEdit.Create constructor as following:

constructor TRxDBComboEdit.Create(AOwner: TComponent);
begin
 {...}
 AlwaysEnable := True; { !! add this line }
end;

Q: Exception 'No SQL statement available' when opening form with TRxQuery that contains no macros and has Active = True.

A: Change RxQuery.pas as following:

procedure TRxQuery.ExpandMacros; var ExpandedSQL: TStringList; begin if not FPatternChanged and not FStreamPatternChanged and (MacroCount = 0) then Exit; **Q**: I'm trying to install RxLib version 2.40 into Delphi 3.0 IDE. Everything goes right, after installing I can use components and compile demo projects. The problem starts when I close Delphi and run again. Then I get error message saying that Delphi was unable to locate a file required for each of the component packages.

•••

A: You must put compiled RX Library run-time packages (DPL files - RXCTL.DPL, RXTOOLS.DPL, RXDB.DPL) into directory that is accessible through the search PATH (i.e. DOS "PATH" environment variable; for example, in the Windows\System directory).

Q: Compilation errors installing RX Library components into Delphi component library. **A:** Perhaps, there are another units or resource files in your Delphi search path (Library Path) named the same as one of the RX Library unit or resource file. Try to rename one of these modules or not to install some of the concurring third-party components (don't forget to remove it from the search path).

Q: Compiler displays an error like "Internal Error ????". **A:** If there is no other component with a similar name or you already have removed such a component and this error still occurs, try disabling Optimization in the compiler settings.

Q: Error "RLINK32: OUT OF MEMORY" during installation RX Library under Delphi 2.0. **A:** There is minor bug in Delphi 2.0 (not 2.01). If you are getting this error while rebuilding your component library with RX components you should recompile resource sources (*.RC) using BRCC32.EXE from your Delphi BIN directory to produce resource files (*.RES and *.DCR) compatible with your version of Delphi 2.

RX Library Help - List by Filename

Class	Brief Description	File
TAnimatedCursorImage	Animated Cursors	<u>AniFile</u>
<u>TAnimateImage</u>	Animation	<u>Animate</u>
<u>TAppEvent</u>	General Purpose Application handler	
Application Units	Visual Application Events (no coding instead)	<u>AppUtils</u>
TDBLocate	Borland Database Engine Utilities	<u>BdeUtils</u>
Box Procedures	Box Procedures	BoxProcs
RX Clip Icons	Code for the clipboard.	<u>ClipIcon</u>
TClipboardViewer	Display the contents of the Windows Clipboard on a form	<u>ClipView</u>
<u>TRxCurrencyEdit</u>	An edit field with currency editing properties	<u>CurrEdit</u>
<u>TConverter</u>	Multiple purpose converter VCL wrapper	DataConv
Date Util	Date Utilities	<u>DateUtil</u>
<u>TDBDateEdit, PickDate</u>	Data Aware version of <u>RX Date Edit</u>	<u>DBCtrl</u>
TRxBdeErrorDlg	Replaces the standard BDE data-aware Error message	DBExcpt
<u>TRxDBFilter</u>	A database filter	<u>DbFilter</u>
TRxDBGrid	Better Grid than some shareware implementations!	
TDBIndexCombo	Provides a drop-down list of the indexes.	
TDBProgress	Monitor a "remote" (or process on another computer) SQL procedure	
TQBEQuery	Query by Example (see your academic notes)	<u>DbQbe</u>
TRxDBRichEdit	Data-Aware RichEdit control	DBRichE
TDBSecurity	Database Security options	
TLocateObject	Database Utilities	<u>DBUtils</u>
TdualListDialog	Allows creating a dual-list box (e.g., installation settings)	<u>DualList</u>
File Utilities	File Utilities	<u>FileUtil</u>
TRxGIFAnimator	GIF Animation Routine	<u>GIFCtrl</u>
<u>TIconList</u>	Internal details and workings of the <u>RX Tray Icon</u> .	<u>IcoList</u>
<u>Min - Max</u>	Minimum and Maximum.	<u>MaxMin</u>
TMemoryTable	In-Line memory table to store temporary queries	<u>MemTabl</u>
<u>TMRUManager</u>	Most Recently Used Manager (e.g., Browser back and forward buttons?)	
<u>THugeList</u>	Huge Object Oriented List	<u>ObjStr</u>
RX Ole 2 Automation	OLE2 Automation stuff.	<u>Ole2Auto</u>
<u>TPageManager</u>	Create wizards	PageMng
TRxMathParser	Mathematical parser	Parsing

Sort by **Component Library**, Filenames or by **Classes**...

<u>TPicClip</u>	Provides the functionality of the PICCLIP.VBX control which ships with Visual Basic Professional	<u>PicClip</u>
<u>TFormPlacement</u>	The TFormPlacement component provides a quick method for saving size and position of its parent form.	<u>Placement</u>
<u>TFormStorage</u>	Stores the form's internal details	<u>Placement</u>
<u>RxGraph</u>	Bitmap to memory formatting	<u>RX Graph</u>
TRxCalculator	Calculator	<u>RxCalc</u>
<u>TRxClock</u>	The standard digit and analogue Clock	<u>RxClock</u>
TRxFontComboBox	TListBox that displays Fonts.	RxCombos
TRxColorComboBox	TListBox that displays a Colour combination.	RxCombos
<u>TRxCalcEdit</u>	Edit field with <u>RxCalculator</u>	RxCombos
<u>TRxTextListBox</u>	This is a TListBox descendant with optional horizontal scrollbar.	<u>RxCtrls</u>
<u>TRxCheckListBox</u>	This displays a scrollable TListBox with check boxes next to each item.	<u>RXCtrls</u>
<u>TRxLabel</u>	TLabel with 3D special effects and other options.	<u>RxCtrls</u>
<u>TRxDrawGrid</u>	An alternative draw grid	<u>RxCtrls</u>
<u>TSecretPanel</u>	Similar to tPanel, except that it can be used as a scroller	<u>RXCtrls</u>
<u>TDBStatusLabel</u>	Information about the status of the database	RxDBCom
TRxDBLookupList	Similar to InfoPower's Woll2Woll components.	RxDBCom
TRxDBComboBox	Similar to InfoPower's Woll2Woll components.	RxDBCom
TRxDBCalcEdit	Data-aware Edit field with <u>RxCalculator</u>	
<u>TRxDice</u>	The dice	<u>RxDice</u>
<u>TGIFFrame</u>	GIF routines	
<u>TRxWindowHook</u>	Allows creating a windows hook (register an event with the windows event system)	
<u>TRxIniFile</u>	The lower level details of <u>RX Form Storage</u> .	<u>RxIni</u>
TRxDBLookupCombo	Similar to InfoPower's Woll2Woll components.	<u>RxLookup</u>
TRxQuery, TSQLScript	Special version of Query and SQL Execute	<u>RxQuery</u>
<u>TRxTrayIcon</u>	Tray Icon - and note the ShellAPI extensions it contains (This is used in combination with <u>AppEvent</u>)	<u>RxShell</u>
<u>TRxSlider</u>	Slider Component	<u>RxSlider</u>
<u>TRxSpeedButton</u>	Speed button with an explorer style alternative.	<u>RxSpeed</u>
<u>TRxSpinEdit</u>	An alternative spinner button	<u>RxSpin</u>
<u>TRxSplitter</u>	Splitter component.	<u>RxSplit</u>
<u>TRxSwitch</u>	The alternative to the Delphi 1.0 VBX switch	<u>RxSwitch</u>
<u>TSpeedbar</u>	A dockable speedbar	<u>Speedbar</u>
RX Splash Windows	Splash window	<u>SplshWnd</u>
TRxStrHolder	A VCL container for stringlist.	<u>StrHolder</u>
INADUITOIUCI		

<u>TRxTimerList</u>	Timer "list of list" (to conserve timer resources)	<u>TimerLst</u>
TrxComboEdit	A combo edit field with a button.	<u>ToolEdit</u>
<u>TRxFilenameEdit</u>	A combo edit field that selects a filename	<u>ToolEdit</u>
<u>TRxDateEdit</u>	An edit field designed for date input and contains a calendar	<u>ToolEdit</u>
TRxDirectoryEdit	An edit field with a pop-up directory selection dialogue box	<u>ToolEdit</u>
RX VCL Utilities	VCL Utilities	<u>VCLUtils</u>
<u>TVerInfo</u>	Version Information Stuff	<u>VerInfo</u>

RX Library Help - List by classes

Class	Brief Description	File
Application Units	Visual Application Events (no coding instead)	<u>AppUtils</u>
Box Procedures	Box Procedures	BoxProcs
<u>Date Util</u>	Date Utilities	<u>DateUtil</u>
File Utilities	File Utilities	<u>FileUtil</u>
<u>Min - Max</u>	Minimum and Maximum.	<u>MaxMin</u>
RX Clip Icons	Code for the clipboard.	<u>ClipIcon</u>
RX Ole 2 Automation	OLE2 Automation stuff.	Ole2Auto
RX Splash Windows	Splash window	<u>SplshWnd</u>
RX StrUtils	String Utilities	<u>StrUtils</u>
RX VCL Utilities	VCL Utilities	<u>VCLUtils</u>
<u>RxGraph</u>	Bitmap to memory formatting	<u>RX Graph</u>
TAnimatedCursorImage	Animated Cursors	<u>AniFile</u>
TAnimateImage	Animation	<u>Animate</u>
<u>TAppEvent</u>	General Purpose Application handler	<u>AppEvent</u>
<u>TClipboardViewer</u>	Display the contents of the Windows Clipboard on a form	<u>ClipView</u>
<u>TConverter</u>	Multiple purpose converter VCL wrapper	DataConv
TDBDateEdit, PickDate	Data Aware version of <u>RX Date Edit</u>	<u>DBCtrl</u>
TDBIndexCombo	Provides a drop-down list of the indexes.	<u>DBIndex</u>
TDBLocate	Borland Database Engine Utilities	<u>BdeUtils</u>
TDBProgress	Monitor a "remote" (or process on another computer) SQL procedure	DBPrgrss
TDBSecurity	Database Security options	<u>DBSecur</u>
TDBStatusLabel	Information about the status of the database	RxDBComb
TdualListDialog	Allows creating a dual-list box (e.g., installation settings)	<u>DualList</u>
<u>TFormPlacement</u>	The TFormPlacement component provides a quick method for saving size and position of its parent form.	<u>Placement</u>
TFormStorage	Stores the form's internal details	<u>Placement</u>
<u>TGIFFrame</u>	GIF routines	<u>RxGIF</u>
<u>THugeList</u>	Huge Object Oriented List	<u>ObjStr</u>
<u>TIconList</u>	Internal details and workings of the <u>RX Tray Icon</u> .	<u>IcoList</u>
<u>TLocateObject</u>	Database Utilities	DBUtils
TMemory Table	In-Line memory table to store temporary queries	<u>MemTable</u>
TMRUManager	Most Recently Used Manager (e.g., Browser back and forward buttons?)	MRUList

Sort by **Component Library**, **Filenames** or by Classes...

<u>TPageManager</u>	Create wizards	PageMngr
<u>TPicClip</u>	Provides the functionality of the PICCLIP.VBX control which ships with Visual Basic Professional	<u>PicClip</u>
<u>TQBEQuery</u>	Query by Example (see your academic notes)	<u>DbQbe</u>
<u>TRxBdeErrorDlg</u>	Replaces the standard BDE data-aware Error message	DBExcpt
<u>TRxCalcEdit</u>	Edit field with <u>RxCalculator</u>	RxCombos
<u>TRxCalculator</u>	Calculator	<u>RxCalc</u>
<u>TRxCheckListBox</u>	This displays a scrollable TListBox with check boxes next to each item.	<u>RXCtrls</u>
<u>TRxClock</u>	The standard digit and analogue Clock	RxClock
TRxColorComboBox	TListBox that displays a Colour combination.	RxCombos
<u>TrxComboEdit</u>	A combo edit field with a button.	<u>ToolEdit</u>
<u>TRxCurrencyEdit</u>	An edit field with currency editing properties	<u>CurrEdit</u>
<u>TRxDateEdit</u>	An edit field designed for date input and contains a calendar	<u>ToolEdit</u>
<u>TRxDBCalcEdit</u>	Data-aware Edit field with <u>RxCalculator</u>	RxDBCtrl
TRxDBComboBox	Similar to InfoPower's Woll2Woll components.	RxDBComb
<u>TRxDBFilter</u>	A database filter	<u>DbFilter</u>
<u>TRxDBGrid</u>	Better Grid than some shareware implementations!	<u>DbGrid</u>
TRxDBLookupCombo	Similar to InfoPower's Woll2Woll components.	<u>RxLookup</u>
<u>TRxDBLookupList</u>	Similar to InfoPower's Woll2Woll components.	RxDBComb
<u>TRxDBRichEdit</u>	Data-Aware RichEdit control	DBRichEd
<u>TRxDice</u>	The dice	<u>RxDice</u>
<u>TRxDirectoryEdit</u>	An edit field with a pop-up directory selection dialogue box	<u>ToolEdit</u>
<u>TRxDrawGrid</u>	An alternative draw grid	<u>RxCtrls</u>
<u>TRxFilenameEdit</u>	A combo edit field that selects a filename	<u>ToolEdit</u>
TRxFontComboBox	TListBox that displays Fonts.	RxCombos
TRxGIFAnimator	GIF Animation Routine	<u>GIFCtrl</u>
<u>TRxIniFile</u>	The lower level details of <u>RX Form Storage</u> .	<u>RxIni</u>
<u>TRxLabel</u>	TLabel with 3D special effects and other options.	<u>RxCtrls</u>
<u>TRxMathParser</u>	Mathematical parser	Parsing
TRxQuery, TSQLScript	Special version of Query and SQL Execute	<u>RxQuery</u>
<u>TRxSlider</u>	Slider Component	<u>RxSlider</u>
TRxSpeedButton	Speed button with an explorer style alternative.	RxSpeed
<u>TRxSpinEdit</u>	An alternative spinner button	<u>RxSpin</u>
<u>TRxSplitter</u>	Splitter component.	<u>RxSplit</u>
<u>TRxStrHolder</u>	A VCL container for stringlist.	<u>StrHolder</u>
<u>TRxSwitch</u>	The alternative to the Delphi 1.0 VBX switch	<u>RxSwitch</u>
<u>TRxTextListBox</u>	This is a TListBox descendant with optional horizontal scrollbar.	<u>RxCtrls</u>
TRxTimerList	Timer "list of list" (to conserve timer resources)	<u>TimerLst</u>

<u>TRxTrayIcon</u>	Tray Icon - and note the ShellAPI extensions it contains (This is used in combination with <u>AppEvent</u>)	<u>RxShell</u>
TRxWindowHook	Allows creating a windows hook (register an event with the windows event system)	<u>RxHook</u>
<u>TSecretPanel</u>	Similar to tPanel, except that it can be used as a scroller	<u>RXCtrls</u>
<u>TSpeedbar</u>	A dockable speedbar	<u>Speedbar</u>
<u>TVerInfo</u>	Version Information Stuff	VerInfo

Control Name	Unit	Class
RX ComboEdit	<u>ToolEdit</u>	TComboEdit

Description:

The TComboEdit component combines an Edit field with a tButton. The button is automatically aligned to the right side of the edit field and triggers an event when clicked (<u>OnButtonClick</u>).

The TComboEdit component is often used as a replacement as it can invoke a dialog box used to provide lookup values.

To change the button's glyph and appearance, use the GlyphKind, Glyph, NumGlyphs, ButtonWidth and ButtonHint properties.

Property Button

Declaration: Button: TRxSpeedButton;

This run-time and read-only property provides to direct access to the button in the button edit component (see also: <u>TRxSpeedButton</u> control.)

Property ButtonHint

Declaration: ButtonHint: string;

The ButtonHint property is the text string that specifies the Hint property for the button in a button edit component (TComboEdit and derived).

Property ButtonWidth

Declaration: ButtonWidth: Integer;

For button edits, use this property to specify the width of the button. Set this property to zero value to hide the button in a combo-edit control.

Event OnButtonClick

Declaration: OnButtonClick: TNotifyEvent;

OnButtonClick event occurs whenever the button in a button edit component gets pressed or the user presses the accelerator key specified by ClickKey property. When ButtonWidth = 0 the OnButtonClick events doesn't occure.

Method DoClick

Declaration: procedure DoClick;

The DoClick method simulates a mouse click, as if the user had clicked a button, executing any code attached to the <u>OnButtonClick</u> event.

Property EditText : string EditText variable contains the text in the edit field. Use this function to programatically obtain the data from the control.

Property Text : string Text property contains the text of the ComboEdit field.

RX Date Edit Component - A edit field with a calandar

Class <u>TDateEdit</u>

Public: Button Button Hint Button Width Check Valid Date Date Default Today Dialogue Title Direct Input Do Click Get Date Mask OnAcceptDate OnButtonClick StartOfWeek WeekEnd WeekEnds

<u>RX ComboEdit component</u> - A combo edit field with a button.

Class: <u>ComboEdit</u>

Public: ButtonHint ButtonWidth DoClick OnButtonClick EditText Text

<u>RX Filename Edit Component</u> - A combo edit field that selects a filename

Class: <u>tFileNameEdit</u>

Public: <u>AcceptFiles,</u> <u>DefaultExt,</u> <u>Dialog,</u> DialogFiles, DialogKind, DialogOptions, EComboEditError, FileName, History, InitialDir, OnAfterDialogue OnBeforeDialog scAltDown TDirDialogKind TExecDateDialog TExecOpenDialogEvent TFileDialogKind TGlyphKind

Control Name	Unit	Class
RX File Edit	<u>ToolEdit</u>	TFilenameEdit

Description:

The TFilenameEdit is a normal TEdit field specially designed for inputting filenames. The TFilenameEdit control has button. The button is automatically aligned to the right side of the edit field and triggers an event when clicked (<u>OnButtonClick</u>).

On the click event of this button you get a dialog to allow the user to select a file name. The standard Delphi TOpenDialog component used to show dialog box. When the user chooses OK in the dialog box, the user's filename selection is stored in the edit box's FileName property, which you can then use to process as you want. The Text property contains same string as the <u>FileName property</u>, but in 32-bit version the long filenames with spaces enclosed in double-quotes.

Property AcceptFiles

Declaration: AcceptFiles: Boolean;

AcceptFiles property specifies whether or not drag-and-drop operations are enabled. When True, user can drag files and directories from the FileManager or Windows Explorer and drop it on the TFilenameEdit or TDirectoryEdit controls. The OnDropFiles event is called when files are dropped on the edit box.

Property Dialog

Declaration: Dialog: TOpenDialog;

Read-only property to the direct access to the TOpenDialog component, than called by the TFilenameEdit control.

Property DialogFiles

Declaration: DialogFiles: TStrings;

Run-time and read only. The DialogFiles property value contains a list of all the file names selected in the TOpenDialog dialog box including the path names.

Property DialogKind

Declaration: DialogKind: TFileDialogKind;

Use this property to select between standard dialog types:

- dkOpen TOpenDialog
- dkSave TSaveDialog
- dkOpenPicture TOpenPictureDialog (Delphi 3.0 or higher)
- dkSavePicture TSavePictureDialog (Delphi 3.0 or higher)

Property DialogOptions

Declaration: DialogOptions: TOpenOptions;

DialogOptions is a set of flags that determine the behavior of the popup dialog. This property allows you to set the Options property of the TOpenDialog component.

Property FileName

Declaration: FileName: string;

The FileName property specifies the file name that the TFilenameEdit displays and that appears in the File Name edit box when the pop-up dialog box opens.

Event OnAfterDialog

Declaration: OnAfterDialog: TExecOpenDialogEvent;

OnAfterDialog event occurs on filename or directory edit controls just after the open-file dialog or selectdirectory dialog was closed. OnAfterDialog allows an application to validate the selected value. The value of the Action parameter determines if the editor can accept new value or not.

Event OnBeforeDialog

Declaration: OnBeforeDialog: TExecOpenDialogEvent;

OnBeforeDialog event occurs on filename or directory edit controls before the open-file dialog or selectdirectory dialog was showed. OnBeforeDialog allows an application to change the value passed to the dialog. The value of the Action parameter determines if the dialog can show or not.

Event OnDropFiles

Declaration: OnDropFiles: TNotifyEvent;

If the AcceptFiles property is True then this event is called when files are dropped on the edit box from FileManager or Windows Explorer.

Const scAltDown

Declaration: scAltDown = scAlt + vk_Down;;

scAltDown constant specifies the default key combination (Alt+Down) for ClickKey property of the TComboEdit, TDateEdit, TFilenameEdit and TDirectoryEdit components.

Type EComboEditError

Declaration: EComboEditError = class(Exception);

EComboEditError is the exception class for data that are invalid for TComboEdit and inherited components.

Type TCalendarStyle

Declaration: TCalendarStyle = (csPopup, csDialog);

The TCalendarStyle type is the type of the CalendarStyle property for a date editors (TDateEdit and TDBDateEdit components).

Type TDirDialogKind

Declaration: TDirDialogKind = (dkVCL, dkWin32);

The TFileDialogKind type contains the values the DialogKind property of directory edit can assume.

Type TExecDateDialog

Declaration: TExecDateDialog = procedure(Sender: TObject; var Date: TDateTime; var Action: Boolean) of object;

The TExecDateDialog type points to a method that handles the retrieving of the date value from pop-up calendar or modal dialog to the TDateEdit control.

Type TExecOpenDialogEvent

Declaration: TExecOpenDialogEvent = procedure(Sender: TObject; var Name: string; var Action: Boolean) of object;

The TExecOpenDialogEvent type points to the method that handles the closing of a pop-up dialog.

Type TFileDialogKind

Declaration: TFileDialogKind = (dkOpen, dkSave, dkOpenPicture, dkSavePicture);

The TFileDialogKind type contains the values the DialogKind property of filename edit can assume. Values dkOpenPicture, dkSavePicture are allowable in Delphi 3.0 or higher only.

Type TGlyphKind

Declaration: TGlyphKind = (gkCustom, gkDefault, gkDropDown, gkEllipsis);

The TGlyphKind type contains the values the GlyphKind property of combo edits (TComboEdit, TFilenameEdit, TDirectoryEdit, TDateEdit, TDBDateEdit) can assume.

Property Initial Dir: String;

InitialDir variable contains the Initial Directory where to select the file from.

Property History : tStringList History property contains a list of previously selected files.

-

Property DefaultExt: string; DefaultExt variable stores the default extension, example: ".doc"

Control Name	Unit	Class
RX TDateEdit	<u>ToolEdit</u>	TDateEdit

Description:

The TDateEdit is a TEdit field designed for date input. The TDateEdit control has a button so that when you click the button (click event) it pops up a calendar dialog form or calendar (according to CalendarStyle property value) so that you can choose a date.

The calendar form/ popup contains navigation controls (arrows) that responds to the keyboard and allows setting the selected date on the calendar to the next or previous month or year.

Other properties include such as <u>DefaultToday</u>, <u>DirectInput</u> (validated) and OnExit (after the focus leaves the ComboBox) properties.

The <u>Date</u> property contains the date value displayed in the Edit box. If the property <u>DefaultToday</u> is set to True and the <u>Date</u> property is zero then the current date is assigned to the <u>Date</u> property.

To control whether the year is two or four digits you use a typed boolean constant <u>FourDigitYear</u> from <u>DateUtil</u> unit.

Event OnAcceptDate

Declaration: OnAcceptDate: TExecDateDialog;

The event OnAcceptDate triggers just after the pop-up calendar (or form) is closed. The OnAcceptDate event allows an application to (optionally) validate the date value. The value of the **Action** parameter (a boolean variable) determines if the application should reject or accept the date specified by the user.

Method CheckValidDate

Declaration: procedure CheckValidDate;

CheckValidDate method checks the date value in the edit control and raises the exception EConvertError if the date in the edit box is invalid or an empty date.

Method GetDateMask

Declaration: function GetDateMask: string;

GetEditMask returns the validation edit mask (or the validation control during data entry). This mask is created according to the current Windows settings and to the <u>FourDigitYears</u> global variable (declared in <u>DateUtil</u> unit).

Property Button

Declaration: Button: TRxSpeedButton;

This run-time and read-only property provides to direct access to the button in the button edit component

(see also: <u>TRxSpeedButton</u> control.)

Property ButtonHint

Declaration: ButtonHint: string;

The ButtonHint property is the text string that specifies the Hint property for the button in a button edit component (<u>TComboEdit</u> and derived).

Property ButtonWidth

Declaration: ButtonWidth: Integer;

This is the width of the button on the right of the DateEdit field. Use this property to specify the width of the button. Set this property to zero value to hide the button in a combo-edit control.

Event OnButtonClick

Declaration: OnButtonClick: TNotifyEvent;

OnButtonClick event occurs whenever the button in a button edit component gets pressed or the user presses the accelerator key specified by ClickKey property. When ButtonWidth = 0 the OnButtonClick events doesn't occur.

Method DoClick

Declaration: procedure DoClick;

The DoClick method simulates a mouse click, as if the user had clicked a button, executing any code attached to the <u>OnButtonClick</u> event.

Property DialogTitle : string

DialogTitle specifies the title of the popup dialog.

Property DefaultToday : boolean

DefaultToday Property specifies whether the date should default as today's date.

Property StartOfWeek : Mon...Sun

StartOfWeek property specifies the starting day (e.g., Monday) for each week.

Property WeekendColour : tColourSet;

WeekendColor specifies the colour the label weekend should be in. (e.g., red colour).

Property Weekends : set

Manipulating weekends property to specify which date should be classified as weekends.

Property Direct Input : boolean

DirectInput property specifies whether the date should be directly input by typing or by selecting the calendar. If the value stored in this variable is true, then the date is directly validated.

Property Date variable Date : tDateTime Variable Datecontains the date which is displayed on the tDateEdit.

Control Name	Unit	Class
RX Directory Edit	<u>ToolEdit</u>	TDirectoryEdit

Description: The TDirectoryEdit is a normal TEdit field for inputting directory names. The TDirectoryEdit control has a button attached to it. When the Onclick event is triggered (when you click on the button), a dialog appears to allow the user to select a directory name. When the user chooses OK in the dialog box, the selected directory name is placed into the edit box's **Text** property.

Optionally, you can also permit the user to choose multiple directory names with the MultipleDir property.

This component use the SelectDirectory (Delphi) standard function from FileCtrl unit to show the dialog box.

Property DialogKind

Declaration: DialogKind: TDirDialogKind;

Use this property to select between standard dialog types:

• dkVCL - standard Delphi VCL dialog (using SelectDirectory standard function);

• dkWin32 - standard Windows "Browse For Folder" dialog, used only in Windows95 or NT 4.x or higher. In NT 3.51 standard Delphi VCL dialog will be used regardless this property value.

NOTE. When you set DialogKind to dkWin32 value, the DialogOptions property will be ignored.

Property DialogOptions

Declaration: DialogOptions: TSelectDirOpts;

The DialogOptions properties is a set of values. If DialogOptions is the empty set, the user can only select a directory that already exists. No edit box is provided for the user to enter a new directory name.

These are the possible values that can be added to the DialogOptions set:

- **sdAllowCreate** - an edit box appears to allow the user to type in the name of a directory that does not exist. This option does not create a directory, but the application can access the Text property to create the directory selected if desired.

- **sdPerformCreate** - Used only when DialogOptions contains sdAllowCreate. If the user enters a directory name that does not exist, the TDirectoryEdit creates it just the dialog is closed.

- **sdPrompt** - Used when DialogOptions contains sdAllowCreate. Displays a message box that informs the user when the entered directory does not exist and asks if the directory should be created. If the user chooses OK, the directory is created if DialogOptions contains sdPerformCreate. If DialogOptions does not contain sdPerformCreate, the directory is not created: the application should create it separately.

NOTE. In 32-bit version when DialogKind = dkWin32 this property is ignored.

Property MultipleDirs Declaration: MultipleDirs: Boolean;

When MultipleDir = True, the directory name selected in the dialog will be added to the Text property; when False, the selected directory name will replace the text of the editor.

Control Name	Unit	Class
Currency Edit	<u>CurrEdit</u>	tCurrencyEdit

Description:

The TCurrencyEdit component is a modified TEdit component. CurrencyEdit allows the user to enter only numeric characters, so the user cannot enter invalid characters. When the user leaves the field, the number is reformatted to display appropriately, accordingly with the DisplayFormat property.

The field value is stored in a Value property so you should read and write to that in your program. This field is of type is **Extended**. You can also read and write Value as integer number using the <u>AsInteger</u> property.

Property AsInteger

Declaration: AsInteger: Longint;

Run-time only. This is a conversion property. For a TCurrencyEdit and TRxCalcEdit, AsInteger can be used to read or set the Value as a Longint.

Property BeepOnError

Declaration: BeepOnError: Boolean;

BeepOnError determines whether a beep is sounded when an invalid character is entered. If BeepOnError is True, MessageBeep is called when an invalid character is entered or when a character is entered and the field is full.

Property DecimalPlaces

Declaration: DecimalPlaces: Word;

DecimalPlaces is the number of digits that are displayed to the right of the decimal point. This property can be used to set a decimal position that will display a fractional portion of the Value only if one exists and the TCurrencyEdit or the TRxCalcEdit component have focus.

Property DisplayFormat

Declaration: DisplayFormat: string;

The DisplayFormat property is used to format the value of the field for display purposes. Formatting is performed by FloatToTextFmt function.

Property DisplayText

Declaration: DisplayText: string;

Run-time and read-only. The string value for the field when it is displayed in the TCurrencyEdit or in the TRxCalcEdit control that is not in edit mode (have not focus).

NOTE. The Text property always returns unformatted string.

Property MaxValue

Declaration: MaxValue: Extended;

Use this property to specify the maximum value that can be entered.

Property MinValue

Declaration: MinValue: Extended;

Use this property to specify the minimum value that can be entered.

Property Value

Declaration: Value: Extended;

Value is the actual data in a TCurrencyEdit or in a TRxCalcEdit. Use Value to read data directly from and write data directly to an editor. When Text property is empty string, the Value property returns 0 (zero).

Property ZeroEmpty

Declaration: ZeroEmpty: Boolean;

When True and Value property has zero value (0) the focused editor has empty Text property (empty string). When False editor always shows '0' string when Value = 0. The default value is True..

Unit CurrEdit / <u>CurrencyEdit</u> Public: AsInteger BeepOnError DecimalPlaces DisplayFormat DisplayText MaxValue MinValue Value Zero Empty

Control Name	Unit	Class
Text List Box	RX Ctrls	TTextListBox

Description:

TTextListBox is a TListBox descendant (for text items only, not owner-drawn types) with an automatic horizontal scrollbar if necessary.

Type TAcceptKeyEvent

Declaration: TAcceptKeyEvent = function (Sender: TObject; var Key: Char): Boolean of object;

TAcceptKeyEvent is the type of the OnAcceptEditKey event of the TRxDrawGrid component.

Type TChangeStateEvent

Declaration: TChangeStateEvent = procedure (Sender: TObject; Index: Integer) of object;

TChangeStateEvent is the type of the OnChangeState event of the TRxCheckListBox component.

Type TCheckKind

Declaration: TCheckKind = (ckCheckBoxes, ckRadioButtons);

The TCheckKind type lists the values the CheckKind property of the check list box (TRxCheckListBox) can assume.

Type TEditAlignEvent

Declaration: TEditAlignEvent = procedure (Sender: TObject; ACol, ARow: Longint; var Alignment: TAlignment) of object;

TEditAlignEvent is the type of the OnGetEditAlign event of the TRxDrawGrid component.

Type TEditLimitEvent

Declaration: TEditLimitEvent = procedure (Sender: TObject; var MaxLength: Integer) of object;

TEditLimitEvent is the type of the OnGetEditLimit event of the TRxDrawGrid component.

Type TEditShowEvent

Declaration: TEditShowEvent = procedure (Sender: TObject; ACol, ARow: Longint; var AllowEdit: Boolean) of object;

TEditLimitEvent is the type of the OnShowEditor event of the TRxDrawGrid component.

The value of the AllowEdit parameter determines if the in-place editor can show or not.

Type TEditStyleEvent

Declaration: TEditStyleEvent = procedure (Sender: TObject; ACol, ARow: Longint; var Style: TInplaceEditStyle) of object;

TEditStyleEvent is the type of the OnGetEditStyle event of the TRxDrawGrid component.

The Style parameter determines how users can edit the data of a grid cell. These are the possible values of Style: - ieSimple - No combo box or ellipsis button is provided. The user cannot select the cell's content from a list.

- iePicklist The grid displays a combo box in the column, and the user can choose a value from the drop-down list. You must specify contents of a pick-list by handling OnGetPicklist event.
- ieEllipsis The column displays an ellipsis button that the user can click to choose a value. Clicking the ellipsis button triggers an OnEditButtonClick event.

Type TGetItemWidthEvent

Declaration: TGetItemWidthEvent = procedure(Control: TWinControl; Index: Integer; var Width: Integer) of object;

TGetItemWidthEvent is the type of the OnGetItemWidth event of the TRxCheckListBox component.

Type TGlyphLayout

Declaration: TGlyphLayout = (glGlyphLeft, glGlyphRight, glGlyphTop, glGlyphBottom);

TGlyphLayout defines the values the GlyphLayout property of a secret panel (TSecretPanel) can assume.

Type TInplaceEditStyle

Declaration: TInplaceEditStyle = (ieSimple, ieEllipsis, iePickList);

TInplaceEditStyle defines the values the Style parameter of OnGetEditStyle event handler can assume. The Style parameter determines how users can edit the data of a grid cell. These are the possible values of Style:

- ieSimple No combo box or ellipsis button is provided. The user cannot select the cell's content from a list.
- iePicklist The grid displays a combo box in the column, and the user can choose a value from the drop-down list. You must specify contents of a pick-list by handling OnGetPicklist event.
- ieEllipsis The column displays an ellipsis button that the user can click to choose a value. Clicking the ellipsis button triggers an OnEditButtonClick event.

Type TPicklistEvent

Declaration: TPicklistEvent = procedure (Sender: TObject; ACol, ARow: Longint; PickList: TStrings) of object;

TPicklistEvent is the type of the OnGetPicklist event of the TRxDrawGrid component.

Type TRxButtonState

Declaration: TRxButtonState = (rbsUp, rbsDisabled, rbsDown, rbsExclusive, rbsInactive);

TRxButtonState defines a range for the state of TRxSpeedButton. For internal using.

Type TRxDropDownMenuPos

Declaration: TRxDropDownMenuPos = (dmpBottom, dmpRight);

TRxDropDownMenuPos defines the values the MenuPosition property of a TRxSpeedButton can assume.

Type TRxNumGlyphs

Declaration: TRxNumGlyphs = 1..5;

The TNumGlyphs type defines the range of values (1-5) the NumGlyphs property of a speed button (TRxSpeedButton) can assume.

Type TShadowPosition

Declaration: TShadowPosition = (spLeftTop, spLeftBottom, spRightBottom, spRightTop);

TShadowPosition is the type of the ShadowPos property of the TRxLabel component.

Type TTextLayout

Declaration: TTextLayout = (tlTop, tlCenter, tlBottom);

TTextLayout specifies the available values for vertical placement of the text in the label.

Unit RXCtrls

Class: CheckListBox

Public: AllowGrayed ApplyState AutoScroll Checked CheckedIndex CheckKind EnabledItem State

Triggers OnClickCheck OnGetItemWidth OnStateChange

Class **RxLabel**

Property: Layout ShadowColor ShadowPos ShadowSize ShowFocus

Event: OnMouseEnter OnMouseLeave

Class RX Text List Box

Property TAcceptKeyEvent TChangeStateEvent TCheckKind TEditAlignEvent TEditLimitEvent TEditShowEvent TGetItemWidthEvent TGlyphLayout TInplaceEditStyle TPicklistEvent TRxButtonState TRxDropDownMenuPos TRxNumGlyphs TShadowPosition TTextLayout

Class RX Draw Grid

Event OnAcceptEditKey **Event OnCancelEdit Event OnColumnSized Event OnEditButtonClick Event OnGetEditAlign Event OnGetEditLimit** Event OnGetEditStyle **Event OnGetPicklist Event OnRowSized Event OnShowEditor Method DrawMasked Method DrawMultiline Method DrawPicture Method DrawStr <u>Method InvalidateCell</u> Method InvalidateRow**

Class RX Secret Panel

Property Active Cycled GlyphLayout Lines TextStyle

Events OnStartPlay OnStopPlay
Control Name	Unit	Class
RX Check List Box	<u>RxCtrls</u>	TRxCheckListBox

The TRxCheckListBox displays a scrollable list with a check box next to each item.

TRxCheckListBox is similar to TListBox, except that each item has a check box or radio button (according to the CheckKind property) next to it.

The custom component editor (a wizard) is used to modify the item list and can set all the details - Add, Delete, List, State, Checked and EnabledItem properties at design-time

Optionally, if the <u>AutoScroll</u> property is set to True, a horizontal scroll bar appears automatically on the control. (This is useful for placing a list of items without the list scrolling off the screen)

Note: The word (*check* [x]) = (*tick* [ü]) and the word (*tick* [ü]) = (*check* [x]). Hence the phrase "I checked your work and I found a few mistakes..."

Property AllowGrayed

Declaration: AllowGrayed: Boolean;

This determines whether the checkbox (on the left side of the listbox) can be in unknown (or "grayed") state.

If AllowGrayed is set to True, each item has three possible states: checked, unchecked, and grayed. If AllowGrayed is set to False, each item has only two possible states: checked and unchecked.

Property AutoScroll

Declaration: AutoScroll: Boolean;

The AutoScroll is a Boolean property specifying whether the horizontal scroll bar optionally appears on the list-box control.

This is useful for displaying a large number of string items. The default value is alway set to **True**.

Property Checked[Index: Integer]

Declaration: Checked[Index: Integer]: Boolean;

This property identifies which items on the list are checked.

For each member of the Item List, the boolean value *Checked* is True if a check mark appears on the item's check box (or radio button). *Checked* corresponds to the *cbChecked* state.

(In other words,

- *Checked* property True is equivalent to the RadioButton's state of cbChecked.

- *Checked* property False is equivalent to the RadioButton's state of cbUnchecked or cbGrayed.)

Property CheckedIndex

Declaration: CheckedIndex: Integer;

The value of the CheckedIndex property is the ordinal number (starts from 0 as the first item) of the checked item in the TRxCheckListBox when the CheckKind property is ckRadioButtons.

If no item is checked or (CheckKind = ckCheckBoxes) the value of CheckedIndex is -1.

Property CheckKind

Declaration: CheckKind: TCheckKind;

The property CheckKind determines the kind of check-marks found in the TRxCheckListBox component. These are the possible values and their meanings:

- ckCheckBoxes like check-boxes, allows multiple ticks on any number of items;
- ckRadioButtons like radio-buttons, allows **one tick** on any **one** item only.

Property EnabledItem[Index: Integer]

Declaration: EnabledItem[Index: Integer]: Boolean;

This indicates whether the items in the list are enabled (i.e. user can check and uncheck these items) or disabled (user cannot do anything with that record on the list)

For each member of the Items array, EnabledItem indicates whether this item is enabled. Any item that is disabled appears in the "disabled font style" and the associated check box cannot be changed.

The Index value is zero based (ordinal) and corresponds to the index used in the Items property. Use this property to enable or disable individual list item in a TRxCheckList component.

Note: If Index is -1, there are no items on the list.

Property State[Index: Integer]

Declaration: State[Index: Integer]: TCheckBoxState;

This indicates the state of the items on the list - whether they are checked or grayed.

For each member of the Items array, State indicates whether its check box is selected (cbChecked), deselected (cbUnchecked), or grayed (cbGrayed).

The cbChecked state corresponds to the Boolean property Checked (In other words,

- *Checked* property True is equivalent to the RadioButton's state of cbChecked.

- Checked property False is equivalent to the RadioButton's state of cbUnchecked or cbGrayed.)

If CheckKind = ckRadioButtons, the user selects a radio button in the list box and any selected item is

automatically unselected.

Event OnClickCheck

Declaration: OnClickCheck: TNotifyEvent;

The trigger OnClickCheck occurs when the user selects or de-selects an item's check box.

Write an OnClickCheck event handler to implement any special processing that should occur when the user checks or unchecks an item.

Event OnGetItemWidth

Declaration: OnGetItemWidth: TGetItemWidthEvent;

This obtains the item's width. (It should be bigger than any item or record on the list.)

Event OnStateChange

Declaration: OnStateChange: TChangeStateEvent;

If any changes are made to the control (by the user - NOT by direct manipulation of the CheckedListBox by programming), this is triggered.

Method ApplyState

Declaration:

procedure ApplyState(AState: TCheckBoxState; EnabledOnly: Boolean);

This updates the property of the CheckedListBox.

In other words,

- if the new state is ckCheckBoxes, nothing occurs, except the state is changed to ckCheckBoxes.

- if the new state is ckRadioButtons, the EnabledOnly boolean variable is used to set whether the check list box is enabled or disabled.

Control Name	Unit	Class
<u>RX Colour Combo Box</u>	RxCombos	TColorComboBox
RX Font Combo Box	RxCombos	TFontComboBox

Class - TColorComboBox - Description:

This component is indented for color selection. It is actually a combo-box which drop-down list contains 16 basic system colors. The available colors are the same as the ones in the Object Inspector's drop-down list of colors for the Color property. <u>ColorValue</u> property contains the value of selected color.

Property TColorComboBox.ColorNames

Declaration: ColorNames: TStrings;

ColorNames provides a way to customize the display color's names for the items in the drop-down list of color combo box. Use the ColorNames property to supply color names of your choosing for the individual color items. Each color has a default english name. ColorNames allow the values of any or all of these default names to be replaced by customized names.

ColorNames is a string list. Each name is a string. The first string in the string list becomes the name for the first item on the list (clBlack). The 16-th string becomes the name for the 16-th item (clWhite).

When specifying names at runtime, enter an empty string (") for any name that should keep the default value. Simply leave the line blank when using the string list property editor of the Object Inspector for the ColorNames property.

Property TColorComboBox.ColorValue

Declaration: ColorValue: TColor;

This property represents the color value that currently selected. If you assign a TColor value to ColorValue, the corresponding entry in the list is selected. If the specified TColor is not in the color combo box list, the assignment is ignored and the current selection is unchanged.

Property TColorComboBox.DisplayNames

Declaration: DisplayNames: Boolean;

If True the color names (Black, Red etc) will be displaying in color list to the right of the colors in the color combo box.

Class TFontComboBox

This component is indended for font selection. Its drop-down list contains the list of fonts currently available. Device and Options properties controls what fonts are displayed.

Property TFontComboBox.Device

Declaration: Device: TFontDevice;

This property controls the list of available fonts.

- fdScreen the list will be populated by the screen fonts only.
- fdPrinter the list will be populated by the printer fonts only.
- fdBoth the list will be populated both screen and printer fonts.

Property TFontComboBox.FontName

Declaration: FontName: string;

Contains the name of the selected font.

Property TFontComboBox.Options

Declaration: Options: TFontListOptions;

These are the possible values that can be included in the Options set for the TFontComboBox:

- foAnsiOnly If True, the user can select fonts that use the Windows character set only; that is, the user can't choose a font that contains only symbols because they aren't displayed in the Font combo box.
- foTrueTypeOnly If True, only TrueType fonts are displayed in the Font combo box.
- foFixedPitchOnly If True, only monospaced fonts are displayed in the Font combo box.
- foNoOEMFonts If True, only fonts that aren't vector fonts are displayed in the Font combo box.
- foOEMFontsOnly If True, only OEM fonts are displayed in the Font combo box.
- foScalableOnly If True, only fonts that can be scaled are displayed in the Font combo box.

The default value is [].

Property TFontComboBox.TrueTypeOnly

Declaration: TrueTypeOnly: Boolean;

See Options.

Type TFontComboBox.TFontDevice

Declaration: TFontDevice = (fdScreen, fdPrinter, fdBoth);

The TFontDevice type lists the values the Device property of the Font combo box (TFontComboBox) can assume.

Type TFontComboBox.TFontListOption

Declaration: TFontListOption = (foAnsiOnly, foTrueTypeOnly, foFixedPitchOnly, foNoOEMFonts, foOEMFontsOnly, foScalableOnly);

The TFontListOption type is the set of values the Options property of the Font combo box (TFontComboBox) can have.

Type TFontComboBox.TFontListOptions

Declaration: TFontListOptions = set of TFontListOption;

TFontListOptions is the type of the Options property of the TFontComboBox component.

Unit <u>RX Combos</u>

Class <u>TColorComboBox</u> Property: <u>ColorNames</u> <u>ColorValue</u> <u>DisplayNames</u>

Class **TFontComboBox**

Property Device FontName Options TFontDevice TFontListOption TFontListOptions TrueTypeOnly

Class <u>RxCalcEdit</u>

Property <u>GlyphKind</u> <u>Value</u>

Control Name	Unit	Class
RX Splitter	RxSplit	TRxSplitter

TRxSplitter divides the client area of a form into resizable panes. The splitter (or divisor) is between a control aligned to one of the edges of the form and the controls that fill up the rest of the client area. Give the splitter the same alignment as the control that is anchored to the edge of the form.

When the user moves the splitter, it resizes the anchored control. This, in turn, changes the client area of the form, and the controls that fill up the rest of the client area resize accordingly.

Use each control (ControlFirst ~ ControlSecond) on the form as a separate pane. After each pane is placed, place a splitter with the same alignment to allow that pane (ControlFirst) to be resized. The last pane to be placed on the form (ControlSecond) should be client-aligned, so that it resizes automatically to fill up the remaining space after all other panes are resized.

Using class TRxSplitter

Add a splitter to a form between two aligned controls to allow users to resize the controls at runtime.

Property BottomRightLimit

Declaration: BottomRightLimit: Integer;

TopLeftLimit and BottomRightLimit are the minimum sizes, in pixels, of the client area the splitter must leave on the form. Set TopLeftLimit or BottomRightLimit to provide a minimum size the splitter must leave when resizing its neighboring control. For example, if the Align property is alLeft or alRight, TopLeftLimit is the minimum width of the client area to the left of the splitter and BottomRightLimit is the minimum width of the client area to the right of the splitter. If the Align property is alTop, TopLeftLimit is the minimum height of the client area above the splitter.

Property ControlFirst

Declaration: ControlFirst: TControl;

ControlFirst specifies a first control (left or top), which can be resized by splitter. You need assign value to this property only when automatically assigned value are missed. You can also leave this property blank when ControlFirst value specified.

Property ControlSecond

Declaration: ControlSecond: TControl;

ControlSecond specifies a second control (right or bottom), which can be resized by splitter.

You need assign value to this property only when automatically assigned value are missed. You can also leave this property blank when ControlSecond value specified.

Property TopLeftLimit

Declaration: TopLeftLimit: Integer;

TopLeftLimit and BottomRightLimit are the minimum sizes, in pixels, of the client area the splitter must leave on the form. Set TopLeftLimit or BottomRightLimit to provide a minimum size the splitter must leave when resizing its neighboring control. For example, if the Align property is alLeft or alRight, TopLeftLimit is the minimum width of the client area to the left of the splitter and BottomRightLimit is the minimum width of the client area to the right of the splitter. If the Align property is alTop, TopLeftLimit is the minimum height of the client area above the splitter.

Event OnPosChanged

Declaration: OnPosChanged: TNotifyEvent;

OnPosChanged occurs after the user has retiled the form using the splitter. Write an OnPosChanged event handler to take specific action when the user finishes dragging the edge of the neighboring control using the splitter.

Type TSplitterStyle

Declaration: TSplitterStyle = (spUnknown, spHorizontalFirst, spHorizontalSecond, spVerticalFirst, spVerticalSecond);

TSplitterStyle type defines the values, internally used by the TRxSplitter component.

Unit <u>RXSplitter</u>

Properties BottomRightLimit ControlFirst ControlSecond TopLeftLimit

Events: OnPosChanged

Types: <u>TSplitterStyle</u>

Control Name	Unit	Class
Slider	<u>RxSlider</u>	TRxSlider

This is a slider component that emulates the Win32 slider and optionally-ticked bar control that contains a slider which marks a current Value.

Move the slider's thumb* to the desired value on the bar by dragging it with the mouse or by clicking the mouse on the bar.

(* the "v" thing on the RxSlider --> |----|-v---|---|)

To use the keyboard to move the slider, press the arrow keys or the PgUp and PgDn keys.

The **MaxValue** property sets the maximum position of a TRxSlider. The MinValue property sets the minimum position of a TRxSlider. You can customize the TRxSlider component by using the Orientation property and ImageHThumb, ImageHRuler, ImageVThumb, ImageVRuler properties.

Property EdgeSize

Declaration: EdgeSize: Integer;

"Edge Size".

It is the size of the outer edge of the slider control.

Property ImageHRuler

Declaration: ImageHRuler: TBitmap;

This is the bitmap for the horizontal ruler. In other words, it's the "---" thing on the slider --> |----|-v---|---|)

Property ImageHThumb

Declaration: ImageHThumb: TBitmap;

This is the bitmap for the horizontal ruler. In other words, it's the "v" thing on the slider --> |----|-v---|---|)

Property ImageVRuler

Declaration: ImageVRuler: TBitmap;

This is the bitmap for the vertical ruler. In other words, it's the "---" thing on the slider --> |----|-v---|-v---|)

Property ImageVThumb

Declaration: ImageVThumb: TBitmap;

This is the bitmap for the horizontal ruler. In other words, it's the "v" thing on the slider --> |----|-v---|-v---|)

Property Increment

Declaration: Increment: Longint;

The property Increment is the amount the Value changes when the user presses the arrow keys.

(In other words, if the increment is set to 10, the value changes by +/- 10 per change.)

Property MaxValue

Declaration: MaxValue: Longint;

MaxValue is the maximum Value the thumb of the TRxSlider can move upto.

Use MaxValue to set an upper limit to the Value that can be represented using the slider. The thumb of the slider indicates the current Value in a range between MinValue and MaxValue.

Property MinValue

Declaration: MinValue: Longint;

MinValue specifies the minimum Value of a TRxSlider. Use MinValue to set an lower limit to the Value that can be represented using the slider. The thumb of the slider indicates the current Value in a range between MinValue and MaxValue.

Property NumThumbStates

Declaration: NumThumbStates: TNumThumbStates;

This is the number of thumb states. This allows the flexibility of using a double image or a single image.

```
In other words:

disabled thumb = v

enabled thumb = v

The bitmap is vv.

NumThumbStates = 1, Enabled = True --> |----|-vv---|---|)

NumThumbStates = 1, Enabled = False --> |----|-vv---|---|)

The bitmap is vv.

NumThumbStates = 2, Enabled = True --> |----|-v---|-v---|---|)

NumThumbStates = 2, Enabled = False --> |----|-v---|-v---|---|)
```

Property Options

Declaration: Options: TSliderOptions;

These are the possible values that can be included in the Options set for the TRxSlider control:

- **soSmooth** determines whether the thumb are moved smooth or step-by-step when the user move mouse pointer;
- **soShowPoints** determines whether tick marks are displayed along the track.
- **soShowFocus** indicates whether TRxSlider should draw a focus rectangle around self when it has input focus.

Property Orientation

Declaration: Orientation: TSliderOrientation;

The orientation specifies whether the slider bar is horizontal (soHorizontal) or vertical (soVertical).

Property Value

Declaration: Value: Longint;

The property *Value* contains the current position of the slider of a TRxSlider control.

Value is always in the range between MinValue and MaxValue (inclusive). Set Value to programmatically move the slider to a new value.

Event OnDrawPoints

Declaration: OnDrawPoints: TNotifyEvent;

This event is triggered when the points are drawn.

(This is used where another thing is drawn (e.g., attaching a needle) or to counts the number of points on the slider)

Method DefaultDrawPoints

Declaration: procedure DefaultDrawPoints(PointsStep, PointsHeight, ExtremePointsHeight: Integer); virtual;

This allows the user to draw the points of the RxSlider instead of using the default settings. In other words, the programmer draws the "|" thing on the slider --> |----|--v--|---|)

Example, draw a blue notch for every 5 points.

Type TNumThumbStates

Declaration: TNumThumbStates = 1..2;

TNumThumbStates is the type for the NumThumbStates property of TRxSlider component.

Type TSliderOption

Declaration: TSliderOption = (soShowFocus, soShowPoints, soSmooth);

TSliderOption is a set of values for the Options property of the TRxSlider component.

Type TSliderOptions

Declaration: TSliderOptions = set of TSliderOption;

TSliderOptions is a set of values for the Options property of the TRxSlider component.

Type TSliderOrientation

Declaration: TSliderOrientation = (soHorizontal, soVertical);

TSliderOrientation defines the values for the Orientation property of TRxSlider. The possible values are soHorizontal (default, slider is horizontal) or soVertical (slider is vertical).

Unit RxSlider / <u>RX Slider</u>.

Property DefaultDrawPoints EdgeSize ImageHRuler ImageHThumb ImageVRuler Increment MaxValue MinValue NumThumbStates Options Orientation Value

Types TNumThumbStates TSliderOption TSliderOptions TSliderOrientation

Events: OnDrawPoints

Control Name	Unit	Class
RxLabel	<u>RXCtrl</u>	TRxLabel

TRxLabel is TLabel with some additional special effects. It supports vertical alignment and shadows (like 3D style), in addition to standard label properties. You can use <u>ShadowPos</u>, <u>ShadowColor</u> and <u>ShadowSize</u> properties to determines how to draw the label shadowed.

Property Layout

Declaration: Layout: TTextLayout;

The property *Layout* specifies the vertical placement of the text within the label when the AutoSize property is False. Set Layout to specify how the text of the label is placed within the ClientRect of the label control. Layout can be one of the following values:

- tlTop The text appears at the top of the label.
- tlCenter The text is vertically centered in the label.
- tlBottom The text appears along the bottom of the label.

Property ShadowColor

Declaration: ShadowColor: TColor;

Use this property to specify the color of the shadow. This property is only applicable when ShadowSize is set to non-zero value.

Property ShadowPos

Declaration: ShadowPos: TShadowPosition;

Use this property to specify the position of the shadow. This property is only applicable when ShadowSize is set to non-zero value.

Property ShadowSize

Declaration: ShadowSize: Byte;

Use this property to specify how far down and to the right the shadow text will be displayed from the main text.

Property ShowFocus

Declaration: ShowFocus: Boolean;

ShowFocus indicates whether TRxLabel should draw a focus rectangle around the self ClientRect when the

control specified by FocusControl property has input focus.

Event OnMouseEnter

Declaration: OnMouseEnter: TNotifyEvent;

An OnMouseEnter event occurs when the user enter mouse pointer to the control. You can use OnMouseEnter and OnMouseLeave event handlers to change color, font, style of control etc.

Event OnMouseLeave

Declaration: OnMouseLeave: TNotifyEvent;

An OnMouseLeave event occurs when the user takes away mouse pointer from the control. You can use OnMouseEnter and OnMouseLeave event handlers to change color, font, style of control etc.

Control Name	Unit	Class
RX Clock	RxClock	TRxClock

The TRxClock component allows a standard analog or digital clock (according to the ShowMode property).

The TRxClock can also be used as an *alarm clock* by setting the <u>AlarmEnabled</u>, <u>AlarmHour</u>, <u>AlarmMinute</u> and <u>AlarmSecond</u> properties and writing a handler for the <u>OnAlarm</u> event to trigger.

Property AlarmEnabled

Declaration: AlarmEnabled: Boolean;

This property determines whether the alarm should be activated (trigger) or not.

Property AlarmHour

Declaration: AlarmHour: Byte;

This property defines the time (with the <u>AlarmMinute</u> and <u>AlarmSecond</u> properties) to be used as alarm time checks by the TRxClock component. When computer's time is equal to the time specified by these properties, and <u>AlarmEnabled</u> is True, the <u>OnAlarm</u> event is occured. The hour value uses a 24 hour clock.

Property AlarmMinute

Declaration: AlarmMinute: Byte;

This property defines the time (with the <u>AlarmHour</u> and <u>AlarmSecond</u> properties) to be used as alarm time checks by the TRxClock component.

When computer's time is equal to the time specified by these properties, and <u>AlarmEnabled</u> is True, the <u>OnAlarm</u> event is occured.

Property AlarmSecond

Declaration: AlarmSecond: Byte;

This property defines the time (with the <u>AlarmHour</u> and <u>AlarmMinute</u> properties) to be used as alarm time checks by the TRxClock component. When computer's time is equal to the time specified by these properties, and <u>AlarmEnabled</u> is True, the <u>OnAlarm</u> event is occured.

Property AutoSize

Declaration: AutoSize: Boolean;

If the <u>ShowMode</u> is scDigital and the AutoSize property is true, the size of the font of the digital clock automatically changes according to size of clock control.

Property DotsColor

Declaration: DotsColor: TColor;

The DotsColor specifies the color of the hour dots in the TRxClock control when <u>ShowMode</u> is scAnalog. (This is used only when Ctl3D property is True.)

Property LeadingZero

Declaration: LeadingZero: Boolean;

Specifies whether or not single digit hour values have a leading zero if <u>ShowMode</u> property is scDigital.

Property ShowMode

Declaration: ShowMode: TShowClock;

The ShowMode property determines type of clock to display - (*scDigital* = digital), (scAnalog = traditional analogue clock).

Property ShowSeconds

Declaration: ShowSeconds: Boolean;

The ShowSeconds property determines whether or not seconds (the seconds hand in the analogue clock or seconds time in the digital font) will be displayed.

Property TwelveHour

Declaration: TwelveHour: Boolean;

This specifies whether or not TRxClock displays time values using a twelve-hour clock.

In other words, Property TwelveHour = true --> 8:21 p.m. --> "08:21 PM"; Property TwelveHour = false --> 8:21 p.m. --> "20:21 PM";

The values of TimeAMString and TimePMString standard global variables are used during the formatting of the twelve-hour time.

Event OnAlarm

Declaration: OnAlarm: TNotifyEvent;

This event (trigger) is called when there is an alarm.

The OnAlarm event is used to execute code at time specified by <u>AlarmHour</u>, <u>AlarmMinute</u> and <u>AlarmSecond</u> properties. This event occured only if <u>AlarmEnabled</u> property is set to True value.

Method SetAlarmTime

Declaration: procedure SetAlarmTime(AlarmTime: TDateTime);

This allows you to set the alarm time. (Note: The date is *ignored*.)

Type TShowClock

Declaration: TShowClock = (scDigital, scAnalog);

The TShowClock type defines the possible values for the <u>ShowMode</u> property of the TRxClock component.

Unit <u>RxClock</u>

Property: AlarmEnabled AlarmHour AlarmMinute AlarmSecond AutoSize DotsColor LeadingZero ShowMode ShowSeconds TwelveHour

Method: <u>SetAlarmTime</u>

Events: <u>OnAlarm</u>

Property: TShowClock

Control Name	Unit	Class
RX Draw Grid	<u>RxCtrls</u>	TRxDrawGrid

Rx Draw is an draw grid. It allows more complex plug-ins to be activated.

Event OnAcceptEditKey

Declaration: OnAcceptEditKey: TAcceptKeyEvent;

The OnAcceptEditKey event occurs when a user presses a single character key in the in-place editor. OnAcceptEditKey provides a simple screening of keyboard input while the user is editing the contents of a cell. Write an OnAcceptEditKey event handler to take specific action whenever a user presses a character key and to determine whether keys typed by the user are valid for the current grid cell. OnAcceptEditKey does not occur unless the Options property includes goEditing.

Event OnCancelEdit

Declaration: OnCancelEdit: TNotifyEvent;

OnCancelEdit occurs whenever the in-place editor requests to cancel changes to the current cell. This event is called when user press "Escape" key in in-place editor.

Event OnColumnSized

Declaration: OnColumnSized: TNotifyEvent;

OnColumnSized occurs immediately after the width of a column changes. Write an OnColumnSized event handler to provide special processing whenever a column in the grid changes width. Columns can be sized programmatically or by user manipulation. OnColumnSized does not occur unless the Options property includes goColSizing.

Event OnEditButtonClick

Declaration: OnEditButtonClick: TNotifyEvent;

OnEditButtonClick occurs when the user presses the ellipsis button in a grid cell. Write an OnEditButtonClick event handler to bring up an appropriate list or dialog when the user presses the ellipsis button in a grid cell. The OnEditButtonClick event handler can set the value assotiated with the cell based on the user response to the list or dialog. Use Col and Row properties to access the data assotiated with the current grid cell.

Event OnEditChange

Declaration: OnEditChange: TNotifyEvent;

OnEditChange occurs whenever the text for the in-place editor in TRxDrawGrid component may have changed. Write an OnEditChange event handler to take specific action whenever the text for the in-place

editor may have changed. This event provides the first opportunity to respond to modifications that the user types into the in-place editor.

Event OnGetEditAlign

Declaration: OnGetEditAlign: TEditAlignEvent;

OnGetEditAlign occurs when the in-place editor requests an edit alignment. Write an OnGetEditAlign event handler to provide the in-place editor with an alignment that describes the way the text is formatted by the in-place editor. Set the Alignment parameter to the value that describes the edit alignment for the cell specified by the ACol and ARow parameters. OnGetEditAlign does not occur unless the Options property includes goEditing.

Event OnGetEditLimit

Declaration: OnGetEditLimit: TEditLimitEvent;

OnGetEditLimit occurs when the in-place editor requests the number of characters that can be typed into cells. Write an OnGetEditLimit event handler to determine if a limit should be placed on the number of characters a user can type into a cell. Set the MaxLength parameter to the value that describes the number of characters that can be typed into cells. A value of 0 indicates that there is no application-defined limit on the length. OnGetEditLimit does not occur unless the Options property includes goEditing.

Event OnGetEditStyle

Declaration: OnGetEditStyle: TEditStyleEvent;

OnGetEditStyle occurs when the in-place editor requests an edit style. The Style parameter determines how users can edit the data of a grid cell. These are the possible values of Style:

- ieSimple No combo box or ellipsis button is provided. The user cannot select the cell's content from a list.
- iePicklist The grid displays a combo box in the column, and the user can choose a value from the drop-down list. You must specify contents of a pick-list by handling OnGetPicklist event.
- ieEllipsis The column displays an ellipsis button that the user can click to choose a value. Clicking the ellipsis button triggers an OnEditButtonClick event.

OnGetEditStyle does not occur unless the Options property includes goEditing.

Event OnGetPicklist

Declaration: OnGetPicklist: TPicklistEvent;

OnGetPicklist occurs when the in-place editor requests a pick-list for its drop-down control. You must fill contents of string-list specified by PickList parameter by adding lines (and possible objects) to the string-list. OnGetPicklist does not occur unless the Options property includes goEditing and your custom OnGetEditStyle event handler set Style parameter to iePicklist.

Event OnRowSized

Declaration: OnRowSized: TNotifyEvent;

OnRowSized occurs immediately after the height of a row changes. Write an OnRowSized event handler to provide special processing whenever a row in the grid changes height. Rows can be sized programmatically or by user manipulation. OnRowSized does not occur unless the Options property includes goRowSizing.

Event OnShowEditor

Declaration: OnShowEditor: TEditShowEvent;

OnShowEditor occurs before creating the editor for a cell. Write an OnShowEditor event handler to indicate whether the in-place edit control can be created to allow editing. An OnEditShow event handler contains a boolean AllowEdit variable that determines whether a cell is allowed to editing. Its default value is True.

Method DrawMasked

Declaration: procedure DrawMasked(ARect: TRect; Graphic: TBitmap);

This draws an edit mask, or mask around the selected region.

Method DrawMultiline

Declaration: procedure DrawMultiline(ARect: TRect; const S: string; Align: TAlignment);

DrawMultiline draws a string inside the grid's cell specified by the ACol and ARow parameters. The string will be written using the current value of Font property. Align parameter specifies how text is aligned within the cell. Lines are automatically broken between words if a word would extend past the edge of the cell's rectangle. You can use this method in your custom OnDrawCell event handler.

Method DrawPicture Declaration:

procedure DrawPicture(ARect: TRect; Graphic: TGraphic);

This method draws a picture in the selected region.

Method DrawStr

Declaration: procedure DrawStr(ARect: TRect; const S: string; Align: TAlignment);

DrawStr draws a string inside the grid's cell specified by the ACol and ARow parameters. The string will be written using the current value of Font property. Align parameter specifies how text is aligned within the cell. DrawStr displays text on a single line only. To display multiline text in a grid's cell use DrawMultiline method. You can use this method in your custom OnDrawCell event handler.

Method InvalidateCell

Declaration: procedure InvalidateCell(ACol, ARow: Longint);

The InvalidateCell method causes the visible portion of a grid cell to repaint. Nothing is repainted if the cell is not visible on screen. The ACol and ARow parameters specify the column and row of the cell.

Method InvalidateRow

Declaration: procedure InvalidateRow(ARow: Longint);

InvalidateRow invalidates the region occupied by a row so that it will be repainted with the next Windows paint message. Call InvalidateRow when a row changes so that the area it occupies will need to be repainted. The ARow parameter is the row index.

Control Name	Unit	Class
RX Speed Button	RxSpeed	TRxSpeedButton

TRxSpeedButton is similar to the standard TSpeedButton control. This component has the all properties, methods, and events that apply to the TSpeedButton controls, *and* additional properties.

The <u>Flat</u> property determines whether the button has a 3D border that provides a raised or lowered look when the button is unselected. The <u>Transparent</u> property specifies whether the background of the button obscures objects below the button control when the button is drawn. The <u>GrayedInactive</u> property specifies whether the button's glyph will be "grayed" when the button is unselected.

When it has not the mouse focus, the associated bitmap is drawn in shades (actually the bitmap is shaded with clWhite, clBtnHighlight, clBtnShadow and clBlack, so the look remains consistent when you change the system colors) - you can still give a specific non-focus bitmap if you want to.

When the <u>AllowTimer</u> property is set to True, instead of generating a single OnClick event when the button is pressed, TRxSpeedButton continues to generate OnClick events as long as the button is in the down state, with RepeatInterval interval.

Additionally, you can specify the "drop-down" menu for the button by the DropDownMenu property.

Property Alignment

Declaration: Alignment: TAlignment;

Alignment specifies how caption text is aligned within the button (TRxSpeedButton).

Property AllowTimer

Declaration: AllowTimer: Boolean;

When the AllowTimer property is set to True, instead of generating a single OnClick event when the button is pressed, TRxSpeedButton continues to generate OnClick events as long as the button is in the down state, with RepeatPause interval.

Property DropDownMenu

Declaration: DropDownMenu: TPopupMenu;

Specifies a TPopupMenu that can be linked with the button. If <u>MarkDropDown</u> set to true, a small icon is displayed at the right of the bitmap or the caption.

Property Flat

Declaration: Flat: Boolean;

Flat determines whether the button has a a 3D border that provides a raised or lowered look. Set Flat to True

to remove the raised border when the button is unselected and the lowered border when the button is clicked or selected. When Flat is True, use separate bitmaps for the different button states to provide visual feedback to the user about the button state. When the button has not the mouse focus, and Flat and GrayedInactive properties both set to True, the associated bitmap is drawn in shades.

Property GrayedInactive

Declaration: GrayedInactive: Boolean;

This property controls the type of shading when no non-focus bitmap is defined and Flat property is set to True. When the button has not the mouse focus, the associated bitmap is drawn in shades (actually the bitmap is shaded with clWhite, clBtnHighlight, clBtnShadow and clBlack. You can still give a specific non-focus bitmap if you want to by place into Glyph property the bitmap with 5 images (Normal, Disabled, Down, Stay Down, Inactive). The 5th image will be used to draw non-focus bitmap.

When <u>GrayedInactive</u> is True and Glyph contains less then 5 images, the non-focus bitmap will be create for you automatically.

Property InitPause

Declaration: InitPause: Word;

Use this property to specify the number of milliseconds before the first OnClick event is generated after a TRxSpeedButton is depressed when the <u>AllowTimer</u> property is True.

Property MarkDropDown

Declaration: MarkDropDown: Boolean;

If MarkDropDown set to true, a small icon is displayed at the right of the bitmap or the caption when a drop-down menu is linked to the button (DropDownMenu is set).

Property MenuPosition

Declaration: MenuPosition: TRxDropDownMenuPos;

This property defines the menu position, whether it should be drop-down or on the right of the button.

Property ModalResult

Declaration: ModalResult: TModalResult;

This determines whether and how the button closes its (modal) parent form. Setting the TRxSpeedButton component's ModalResult property is an easy way to make clicking the button close a modal form. When a button is clicked, the ModalResult property of its parent form is set to the same value as the button's ModalResult property.

Property RepeatInterval

Declaration: RepeatInterval: Word;

Use this property to specify the interval (in milliseconds) between OnClick events generated when a TRxSpeedButton is depressed and <u>AllowTimer</u> property is True.

Property Transparent

Declaration: Transparent: Boolean;

Transparent specifies whether the background of the button obscures objects below the TRxSpeedButton object when the button is drawn. Default value is False.

_

Method ButtonClick

Declaration: procedure ButtonClick;

The ButtonClick method simulates a mouse click, as if the user had clicked the TRxSpeedButton.

Method CheckBtnMenuDropDown

Declaration: function CheckBtnMenuDropDown: Boolean;

CheckBtnMenuDropDown allows applications to programatically display a TRxSpeedButton's or TSpeedItem's drop-down menu.

Unit Rx Speed

Methods: <u>ButtonClick</u> <u>CheckBtnMenuDropDown</u>

Properties: Alignment AllowTimer DropDownMenu Flat GrayedInactive InitPause MarkDropDown MenuPosition ModalResult RepeatInterval Transparent

Control Name	Unit	Class
RX Spin Edit	<u>RxSpin</u>	TRxSpinButton

Use the TRxSpinButton to add an up-down control to a form, which consist of a pair of arrow buttons, such as the arrows that appear in a spin box. Spin buttons allow users, for example, to change the size of a numerical value by clicking on arrow buttons.

The TRxSpinEdit component combines a edit field with two buttons forming a spinner.

The edit portion of the TRxSpinEdit component can only accept numeric data and the buttons allow the user to increment and decrement the value.

This component can accept integer values as well as **floating point values**. To only allow **integers**, set the (<u>ValueType</u> = vtInteger). If you are using floating point values, use the decimal property to specify the number of decimal places used by a spin edit when displaying floating point values (<u>ValueType</u> = vtFloat).

The <u>Increment</u> property is used to specify the amount the <u>Value</u> is incremented or decremented when one of the buttons is pressed. Increment can be a floating point value. (In other words, if the increment property is 10, the value is incremented or decremented +/- 10).

Property Down : readonly

Declaration: Down: TSpinButtonState;

The Down property of a spin button determines the current button state. (In other words, if the user has depressed the spin button, this property is set to true)

Property DownGlyph

Declaration: DownGlyph: TBitmap;

Use this property to override the default down arrow bitmap used for a spin button component. (In other words, store your own glyph in this property if you want to use another glyph instead)

Property UpGlyph

Declaration: UpGlyph: TBitmap;

Use this property to override the default up arrow bitmap used for a spin button component. (In other words, store your own glyph in this property if you want to use another glyph instead)

Event OnBottomClick

Declaration: OnBottomClick: TNotifyEvent;

The OnBottomClick event is triggered when the user clicks the *spin buttons* in the spin button control. (In other words, if the user clicks the arrow buttons on the TRxSpinEdit to increment or decrement the spin

button value, this event is triggered)

Event OnTopClick

Declaration: OnTopClick: TNotifyEvent;

The OnTopClick event occurs when the user clicks the top button in the spin button control.

Property Decimal Declaration: Decimal: Integer;

This value specifies the number of decimal places to be utilised.

Property EditorEnabled

Declaration: EditorEnabled: Boolean;

Set this property to True to allow a user to type in a numeric value into the edit portion. When this property is False, only the buttons or arrow keys can be used to modify the Value.

Property Increment

Declaration: Increment: Double;

Use this property to specify the amount the Value property is incremented or decremented by when one of the spin buttons is pressed or when either the up or down arrow key is pressed.

Note that the Increment property can be set to a floating point value, thus allowing non-integer increments.

Property MaxValue

Declaration: MaxValue: Double;

Use this property to specify the maximum Value that can be entered.

Property MinValue

Declaration: MinValue: Double;

Use this property to specify the minimum Value that can be entered.

Property Value

Declaration: Value: Double;

This property contains the contents of the edit portion. It is in floating point format.

Property ValueType

Declaration: ValueType: TValueType;

Set this property to vtInteger to restrict values to Integers and to vtFloat to allow floating values input. If keyboard entry is allowed via EditorEnabled, the user is not allowed to enter a decimal point. Therefore, the Decimal property has no effect when ValueType is vtInteger.

Const InitRepeatPause

Declaration: InitRepeatPause = 400;

Pause before repeat timer (MSec).

Const RepeatPause

Declaration: RepeatPause = 100;;

Pause before hint window displays (MSec).

Type TSpinButtonState

Declaration: TSpinButtonState = (sbNotDown, sbTopDown, sbBottomDown);

The TSpinButtonState type contains the values the Down property of spin buttons (TRxSpinButton) can assume.

Unit SpinEdit Public: Decimal Down DownGlyph EditorEnabled Increment InitRepeatPause MaxValue MinValue MinValue RepeatPause TSpinButtonState UpGlyph Value Value

Events: OnBottomClick OnTopClick

Control Name	Unit	Class
RXControls	RxSwitch	TRxSwitch

RX Switch is a custom control representing a switch. This defines the switch 'ON' and 'OFF' as shown below.

TRxSwitch is a simple component that displays two bitmaps, depending on whether the state is "On" or "Off". The component's <u>StateOn</u> property determines which of the two states is displayed, and its <u>GlyphOn</u> and <u>GlyphOff</u> properties determine bitmaps are used.

Property Glyph On : tBitMap

Set the state of the glyph on. In other words, this is the bitmap that your application would load when the state of the switch is on.

Property Glyph Off : tBitMap

Set the state of the glyph off. In other words, this is the bitmap that your application would load when the state of the switch is off.

Property State On : Boolean

Set the swtichSet the state of the glyph on. In other words, this is the bitmap that your application would load when the state of the switch is on.

Text Position : tTextPos

TTextPos = (tpRight, tpLeft, tpAbove, tpBelow, tpNone);

This sets the state of where the text position is. TTextPos is a set of values for the TextPosition property of the TRxSwitch component.

Toggle Key : TShortCut;

This is the toggle key to [on] or [off] the button.

Event OnOff

Declaration: OnOff: TNotifyEvent;

This event is triggered when the the Switch is switched off..

Event OnOn

Declaration: OnOn: TNotifyEvent;

This event is triggered when the the Switch is switched on.

Method ToggleSwitch

Declaration: procedure ToggleSwitch;

This procedure toggles the switch's state.

Index Page | About | Download Creation Date: 4 Feb 1998 | Last Update: 16 Mar 2000 _

Unit <u>RXSwitch</u> Public: Glyph Off Glyph On OnOff OnOn State On Text Position Toggle Key ToggleSwitch
Control Name	Unit	Class
RXControls	<u>RxDice</u>	tRxDice

Description:

RX Dice is a custom control representing a dice. This has six states (1,2,3,4,5,6). The purpose of this component is to emulate a dice.

Property AutoSize : boolean

AutoSize property sizes the dice to the default size.

Property Colour : tColour

Colour property sets the background colour.

Property Interval : integer

Interval property is the wait state in milliseconds.

Property Rotate : boolean

Rotate property rotates the dice. By setting it to true, it will rotate. When the value is false, the dice is in "stop" mode (or does not move).

Property ShowFocus: Boolean;

Property ShowFocus

ShowFocus indicates whether control should draw a focus rectangle around the self ClientRect when the control has input focus and TabStop = True.

Value : TrxDiceValue;

Declaration: TrxDiceValue = 1..6; TRxDiceValue is the type for the Value property of TRxDice component.

The value (or the variable that contains the value) of the dice. In other words, this is the actual value of the dice.

procedure RandomValue;

This "spins" the dice to a random value. In other words, a random value is assigned to the value property of the TRxDice component.

Unit <u>RxDice</u> Property: <u>AutoSize</u> <u>Colour</u> <u>Interval</u> <u>Rotate</u> <u>ShowFocus</u>

Procedure: <u>RandomValue</u>

Type: <u>TrxDiceValue</u>

Control Name	Unit	Class
RxQuery	<u>RxQuery</u>	TRxQuery

Description:

TRxQuery component is inherited from TQuery component. In addition, an application can supply macros values for dynamic queries with the Macros property and the MacroByName method. Standard Delphi's parameters can replace only column names or data values. Macros allows you change at run-time dinamically any part of SQL text.

Property MacroChar

Declaration: MacroChar: Char;

MacroChar determines character which indicates macros in the SQL query text (default value is '%').

Property MacroCount

Declaration: MacroCount: Word;

Run-time and read-only. The MacroCount property specifies how many entries the TRxQuery has in its Macros array, that is, how many macros the query has. Adding a new item to Macros will automatically increase the value; removing an item will automatically decrease the value.

Property Macros

Declaration: Macros: TParams;

When you enter a query, TRxQuery creates a Macros array for the macros of a dynamic SQL statement. Macros is a zero-based array of TParam objects with an element for each macro (indicated by the leading MacroChar character) in the query; that is, the first macro is Macros[0], the second Macros[1], and so on. The number of macros is specified by MacroCount.

Note: Use the MacroByName method instead of Macros to avoid dependencies on the order of the parameters.

Property Macros example

For example, suppose a TRxQuery component named rxQuery2 has the following statement for its SQL property:

SELECT * FROM ITEMS ORDER BY %ORDER and MacroChar = '%'.

An application could use Macros property and MacroByName method to specify the value of the "ORDER" macro as follows:

rxQuery2.Close; rxQuery2.MacroByName('ORDER').AsString := 'ITEMS.ID'; rxQuery2.Open; rxQuery2.Close; rxQuery2.MacroByName('ORDER').AsString := 'ITEMS.NAME'; rxQuery2.Open; rxQuery2.Close; rxQuery2.MacroByName('ORDER').AsString := 'ITEMS.VENDOR'; rxQuery2.Open;

These three statements would sort the query result by the ID, NAME and VENDOR field correspondingly.

Property OpenStatus

Declaration: OpenStatus: TQueryOpenStatus;

OpenStatus reads the current status of the query component after calling OpenOrExec method. The possible values are those of the TQueryOpenStatus type:

- qsOpened query was successfully executed and result set was returned (i.e. SELECT statement);
- qsExecuted query was successfully executed but result set has not generated (i.e. INSERT, UPDATE, DELETE, or any DDL statement);
- qsFailed error has detected during query execution.

Method ExecDirect

Declaration: ExecDirect

This method is used to immediately prepare and execute a query. Use ExecDirect method to avoid having the SQL statement parsed for parameters (i.e. anything preceded by a colon ':').

In this case, it is being used to create an Oracle trigger, where ":new" refers to the new version of the updated record. Conversely, ":old" refers to the record before the update. Oracle also happens to use ":=" as an assignment operator in triggers.

Method ExpandMacros

Declaration: procedure ExpandMacros;

Replace all macroses in the SQL text by their string values before executing of the query. Called automatically by the Open, Prepare or ExecSQL methods, by must be called manually before ExecDirect or OpenOrExec methods if SQL statements has macroses (MacroCount > 0).

Method IsEmpty

Declaration: function IsEmpty: Boolean;

Checks for a dataset without records.

Method MacroByName

Declaration: function MacroByName(const Value: string): TParam;

The MacroByName method returns the element of the Macros property whose Name property matches Value. Use it to assign values to macroses in a dynamic TRxQuery by their names.

Method MacroByName example

rxQuery1.ParamByName('ORDER').AsString := 'ITEMS.NAME';

Method OpenOrExec

Declaration: procedure OpenOrExec(ChangeLive: Boolean);

If you do not know at design time whether a query will return a result set at run time, use OpenOrExec method to execute query. After calling OpenOrExec you can check OpenStatus property value to determine whether a result set has been returned. If ChangeLive parameter is True and an application requests a live result set, but the SELECT statement syntax does not allow it, the OpenOrExec will try query execution again without request a live result set.

Method OpenOrExec example

```
begin
  {...}
  rxQuery1.SQL := Memo1.Lines;
  if Memo1.Lines.Count = 0 then
    Exit;
  rxQuery1.RequestLive := True;
  rxQuerv1.Params.Clear;
  StartWait;
  try
    rxQuery1.OpenOrExec(True);
  finally
    StopWait;
  end:
  if rxQuery1.OpenStatus = qsExecuted then
    MessageDlg('Query successfully executed.', mtInformation, [mbOk], 0);
  {...}
end;
```

Control Name	Unit	Class
RX SQL Scripts	<u>RxQuery</u>	TSQLScript

Description:

RxSql Scripts is an enhanced version of the SQL.

Property SemicolonTerm

Declaration: SemicolonTerm: Boolean;

Property SemiColon allows the term SemiColon to be input into the SQL statement.

Property SQL

Declaration: SQL: TStrings;

Property SQL example: The following examples show SQL statements used with TSQLScript component:

UPDATE NEXTITEM SET NewItem = 1098; UPDATE NEXTORDER SET NewItem = 1099;

or UPDATE NEXTITEM SET NewItem = 1098 / UPDATE NEXTORD SET NewOrder = 1099

or UPDATE NEXTITEM SET NewItem = 1098; / UPDATE NEXTORD SET NewOrder = 1099;

UPDATE NEXTCUST SET NewCustomer = 1034;

Property Transaction

Declaration: Transaction: Boolean;

Property Transaction defines whether the action should be defined as a transaction.

Event AfterExec

Declaration: AfterExec: TNotifyEvent;

After execution of the SQL, event EventAfterExec is triggered.

Event BeforeExec

Declaration: BeforeExec: TNotifyEvent;

Before execution of the SQL, event EventBeforeExec is triggered.

Event OnScriptError

Declaration: OnScriptError: TScriptErrorEvent;

The OnScriptError event occurs when there was an error during execution of a single SQL statement from SQL script. The Sender parameter is the TSQLScript component that had the problem. The E parameter is an EDatabaseError exception object that contains details about the problem.

The LineNo parameter is a number of line in the script, and StatementNo is a sequence number of single SQL statement that caused the error. Action is a var parameter that lets you control how the error is handled when the event handler finishes.

Method ExecStatement

Declaration: procedure ExecStatement(StatementNo: Integer);

Executes a single SQL-statement specified by StatementNo parameter.

Const DefaultMacroChar

Declaration: DefaultMacroChar = '%';;

Specifies default value for the MacroChar property of the TRxQuery component.

Const DefaultTermChar

Declaration: DefaultTermChar = '/';;

Specifies default value for the Term property of the TSQLScript component.

Type TQueryOpenStatus

Declaration: TQueryOpenStatus = (qsOpened, qsExecuted, qsFailed);

The TQueryOpenStatus type is the type of the OpenStatus property for a TRxQuery component.

Type TScriptAction

Declaration: TScriptAction = (saFail, saAbort, saRetry, saIgnore, saContinue);

The TScriptAction type defines the possible values you can assign to the Action parameter in the OnScriptError event of the TSQLScript component, as described below:

- saFail (The default value of Action.) The error is reported as an exception, as it normally would be.
- saAbort The error is discarded by raising a "silent" EAbort exception.
- saRetry The application goes into a loop, repeating the execution of current (last executed) SQL-

statement until it succeeds or you assign Action another value.

- saIgnore The error is discarded, and sript execution continue from next statement.
- saContinue The error is reported as a dialog box (using Application.HandleException method), and sript execution continue from next statement.

Type TScriptErrorEvent

Declaration: TScriptErrorEvent = procedure(Sender: TObject; E: EDatabaseError; LineNo, StatementNo: Integer; var Action: TScriptAction) of object;

TScriptErrorEvent defines the type of the OnScriptError event handler.

Unit RxQuery

RX SQL Scripts

Property SemicolonTerm SQL Transaction

Event AfterExec BeforeExec OnScriptError

Method ExecStatement

Const <u>DefaultMacroChar</u> <u>DefaultTermChar</u>

Types TQueryOpenStatus TScriptAction TScriptErrorEvent

RX Query

Property <u>MacroChar</u> <u>MacroCount</u> <u>Macros</u> <u>OpenStatus</u>

Method ExecDirect ExpandMacros IsEmpty MacroByName OpenOrExec

Control Name	Unit	Class
RX Memory Table	MemTable	TMemoryTable

Description:

Memory tables are tables created in memory (RAM) and the contents are deleted when you close them. They are much faster and are very useful when you need fast operations on small tables. Memory tables do not support certain features (like referntial integrity, indexes, autoincrement fields and BLOBs).

Only standard (logical) BDE field types are supported by the in-memory table. "Physical" field types are not supported. The table is kept in memory if possible, but it could be swapped to disk if the table becomes too big.

The maximum table size is 512Mb with a maximum record size of 16Kb with a maximum of 1024 fields.

Please note that Logical Auto-increment and BLOB fields are not supported due to their nature of operations.

Using class TMemoryTable

Use TMemoryTable to create a "temporary" table never intended to be written to disk. These tables are created by the application for gathering information that is needed temporarily during processing. These tables can be created only with logical types. These tables do not support indexes.

Before create in-memory table you must specify the table structure by adding fields in Dataset Designer (at design time) or by using FieldDefs property (at run time). To create table you need call the CreateTable or Open method (or assign True value to the Active property).

When the table is closed (by the Close method or assigning Active := False), all data in the memory table will be lost.

The TMemoryTable component provides live access to temporary in-memory BDE tables.

Property EnableDelete

Declaration: EnableDelete: Boolean;

If True (the default), the application can delete the current record by using Delete method.

If False, TMemoryTable won't delete records (it is read-only (aka dBase)).

Property RecNo

Declaration: RecNo: Longint;

The RecNo property Indicates the current record in the TMemoryTable.

Use *RecNo* to determine the record number of the current record in the table. Applications might use this property with *RecordCount* to iterate through all the records in a in-memory table, though typically record

iteration is handled with calls to First, Last, MoveBy, Next, and Prior.

RecNo also can be set to a specific record number to position the cursor on that record. You can also position the cursor on the specified record by using GotoRecord method.

Method BatchMove

Declaration: function BatchMove(ASource: TDataSet; AMode: TBatchMode; ARecordCount: Longint): Longint;

The BatchMove method copies or appends records in the TMemoryTable. ASource is a Table or Query linked to a database table containing the source records. AMode is the copy mode; it can be only batAppend or batCopy.

The ARecordCount parameter is used to control the maximum number of records that will be moved. If zero, all records are moved, beginning with the first record in ASource. If ARecordCount is not zero, a maximum of ARecordCount records will be moved, beginning with the current record. If ARecordCount exceeds the number of records remaining in ASource, no wraparound occurs; the operation is terminated. BatchMove returns the number of records operated on.

Method CopyStructure

Declaration:

procedure CopyStructure(ASource: TDataSet);

This method allows you copy field structure from another dataset ASource to the in-memory table. After copying call Open or set Active property to True to activate in-memory table with new structure.

Method GotoRecord

Declaration: procedure GotoRecord(RecordNo: Longint);

Use this method to position the cursor on record, specified by the sequence number RecordNo. First record in the in-memory table has number 1.

Method SetFieldValues

Declaration: procedure SetFieldValues(const FieldNames: array of string; const Values: array of const);

SetFieldValues enables you to perform updates on entire in-memory table. This method sets all fields passed in the FieldNames parameter to the new field values passed in the Values parameter. The assignment of the elements of Values to fields in the FielsNames is sequential; the first element is assigned to the first field, the second to the second, etc.

Unit <u>MemTable</u>

Method BatchMove Method CopyStructure Method GotoRecord Method SetFieldValues

Property EnableDelete Property RecNo

Control Name	Unit	Class
RX QBE Queries	DBQbe	TQBEQuery

Description:

The non-visual TQBEQuery component allows you to specify Paradox-style Query-By-Example query statements that are used to supply data to one or more of the other visual interface components placed on your form.

Multiple table queries and special queries such as Insert, Delete and ChangeTo queries are also supported. Since TQBEQuery component is inherited from Delphi's TDBDataSet component, the Delphi Fields editor is still available within this component. If you want the normal auxiliary QBE tables written to your hard drive, set the AuxiliaryTables property to True. When defining QBE statements, it's a very good idea to create, fully test and then save your query via the Database Desktop or the native application (Paradox or dBASE for Windows). Then, click the Load button of the QBE string editor to load this saved query file into the editor. These environments make it much easier to create, test and validate the results of a QBE than the services provided in Delphi. If you know more about SQL than Paradox-style QBE, or if your tables are from an SQL database, you might want to consider using the TQuery or TRxQuery component instead.

Using class TQBEQuery

If you receive the error message "Error creating cursor handle" when executing your QBE, it means that your query was unable to be executed. Likely causes are incorrect syntax in your QBE specification, unopened aliases, or missing tables.

Property AuxiliaryTables

Declaration: AuxiliaryTables: Boolean;

This property tells the QBE processor whether or not to create the standard auxiliary tables (keyviol, changed, inserted, etc.). When an auxiliary table is created, it remains on your disk until it is deleted. If an auxiliary table already exists, it will be overwritten without warning. The default value is True.

CAUTION: If you set this property to False, you may not be able to fully check the results of your QBE since no auxiliary tables will be created for manual verification.

Property BlankAsZero

Declaration: BlankAsZero: Boolean;

When True blanks in numeric fields in a table are treated equivalent to a zero. The default value is False.

Property ParamCheck

Declaration: ParamCheck: Boolean;

The ParamCheck property determines whether the query fills the Params array when the SQL or QBE

property is changed at run time. Default is True.

Property QBE

Declaration: QBE: TStrings;

This property holds the actual QBE query statements. After creating a query via the Database Desktop, or from within Paradox for Windows, save the query as a .QBE file (File | Save menu options). You can then load this previously saved file into the Delphi String list editor window for the QBE property by clicking on the Load button and selecting the file you previously saved. Optionally, you can specify the QBE statements interactively. The default value is blank - no QBE defined.

NOTES: The contents of the "ANSWER:" QBE statement generated when you save a query to a file are ignored by the BDE QBE processor. The contents of the Selected Fields list box of the Select Fields dialog box default to all fields that are selected via the QBE statements contained in the QBE property.

Property StartParam

Declaration: StartParam: Char;

StartParam determines character which indicates parameter in the QBE query text (default value is '#').

Method ExecQBE

Declaration: procedure ExecQBE;

Use the ExecQBE method to execute an QBE statement assigned to the QBE property of a TQBEQuery if the statement does not return a result set. If the QBE statement is an INSERT, UPDATE, DELETE, or any DDL statement, then use this method.

Unit **DbQbe**

Using class TQBEQuery Method: ExecQBE

Property AuxiliaryTables BlankAsZero ParamCheck QBE StartParam

Control Name	Unit	Class
RX Database Filters	DBFilter	TRxDBFilter

Description:

This component implements a BDE-level filter, based on a callback into your form's/unit's code. The TRxDBFilter component allows you to specify one or more record selection criteria to be used when displaying data from a table or query (TTable, TQuery, <u>TQBEQuery</u> or <u>TRxQuery</u> components). Filter criteria can be specified for any number of fields in a table, giving you more flexibility than Delphi's built-in Range operators and some advantages over performing a single-table query.

You can specify one or more lines of filter criteria in <u>Filter property</u> to be used when displaying data from the associated table or query. The default value is blank - no filter defined. Applications can also specify a filter using the <u>OnFiltering</u> event handler. The Filter property supplements the OnFiltering event handler. Be sure that the interactions between the Filter property and the OnFiltering event handler do not result in an empty filter set when they are used simultaneously in an application.

Delphi 2.0 and 3.0 TTable and TQuery components has a similar property named Filter, which has similar functionality. However Delphi 2.0 is for 32 bit applications only. We recommend you use Delphi's filtering if you are developing 32-bit only applications, but use TRxDBFilter if you need to simultaneously support both 16 and 32 bit. You can also use same data-aware controls (i.e. TDBEdit, TDBCheckBox etc) to display and edit dataset's records and to enter filter's selection criteria. Use SetCapture, ReleaseCapture and ReadCaptureControls methods to enable users to enter selection criteria into data-aware controls at runtime.

Property Active

Declaration: Active: Boolean;

Specifies whether filtering is active for a dataset. Check Active to determine whether or not dataset filtering is in effect. If Active is True, then filtering is active. Otherwise Active is False. To apply filter conditions specified in the Filter property or the OnFiltering event handler, set Active to True (or call Activate method). When filtering is enabled, user edits to a record may mean that the record no longer meets a filter's test condition. The next time the record is retrieved from the dataset while the filter is in effect, the record may seem to disappear. If that happens, the next record that passes the filter condition becomes the current record.

Property Captured

Declaration: Captured: Boolean;

Captured (runtime and read-only) indicates whether the filter has 'captured' data-aware controls assosiated with filtered dataset (specified by DataSource property).

A data-aware controls becomes the "captured controls" when the TRxDBFilter calls SetCapture method.

To release captured controls application must call the ReleaseCapture method. When data-aware controls are captured by the filter, the dataset's Post method calls the ReadCaptureControls; ReleaseCapture; Activate; methods and sets the filter by using the entered values; the dataset's Cancel method releases controls by calling ReleaseCapture; method.

Property ExprFilter

Declaration: ExprFilter: hDBIFilter;

Specifies the Borland Database Engine (BDE) filter handle for the filter, which use the Filter property.

Property Filter

Declaration: Filter: TStringList;

Specifies the text of the current filter for a dataset. Use Filter to specify a dataset filter. When filtering is applied to a dataset (by setting the Active property to True or calling Activate method), only those records that meet a filter's conditions are available to an application. Filter contains the string that describes the filter condition. For example, the following filter condition displays only those records where the customer number greather then 1000 and customer name starts with 'A' character: ((Cust_No > 1000) OR ([Customer Name] < 'B')) The following conditions are also available: State = 'CA' or Pay = NULL or [Date] > '12/01/1995'. To filter strings bases on partial comparisons, use an asterisk as a wildcard. For example: State = 'M*' The following unary operations available in a filter expressions: = NULL <> NULL

The following binary operations available in a filter expressions:

- <> (not equal)
- >= (great or equal)
- <= (less or equal)
- = (equal)
- > (great)
- < (less)

Applications can set Filter at runtime to change the filtering condition for a dataset at (for example, in response to user input).

Property Filter example

```
with rxDBFilter1.Filter do
begin
   BeginUpdate;
   try
      Clear;
      Add('((Cust_No > 1000) OR');
      Add('([Customer Name] <= "B")) AND (State = "CA")');
   finally
      EndUpdate;
   end;
end;
end;
rxDBFilter1.Active := True;</pre>
```

Property FuncFilter

Declaration: FuncFilter: hDBIFilter;

Specifies the Borland Database Engine (BDE) filter handle for the filter, which use the OnFiltering event handler.

Property LogicCond

Declaration: LogicCond: TFilterLogicCond;

The LogicCond property specifies a logical operator which used to create selection criteria with the ReadCaptureControls method. The valid values for LogicCond are flAND or flOR. If LogicCond is flAND, all simple filter conditions which has been read from data-aware controls, combines with "AND" logical operator. If LogicCond is flOR, the "OR" logical operator used to combine filter conditions into selection criteria (Filter property).

Property Options

Declaration: Options: TDBFilterOptions;

Options lets you fine-tune the filtering provided by the Filter property. The TFilterOptions type defines the possible values for the Options property, as described below:

- foCaseInsensitive - The filter is processed without regard to case in the dataset's data.

- foNoPartialCompare - String matches must be exact over the length of the data in the dataset; partial matches aren't allowed.

Property Priority

Declaration: Priority: Word;

Priority determines the filter's priority relative to other filters on the same dataset.

Event OnActivate

Declaration: OnActivate: TNotifyEvent;

The OnActivate event is activated after a filter is activated, either by calling the Activate method or by setting the Active property to True. By assigning a method to this property, you can take any special actions required by the event.

Event OnDeactivate

Declaration: OnDeactivate: TNotifyEvent;

The OnDeactivate event is activated after a filter is deactivated, either by calling the Deactivate method or by setting the Active property to False.

By assigning a method to this property, you can take any special actions required by the event.

Event OnFiltering

Declaration: OnFiltering: TFilterEvent;

OnFiltering occurs each time a different record in the dataset becomes the current record and filtering is enabled. Write an OnFiltering event handler to test each record in a dataset for against a test condition that determines whether or not the record is visible to the application. To indicate whether or not a record passes the filter condition, a filter handler must return True to include a record, or False to exclude it.

Applications can also specify a filter using the Filter property. The Filter property supplements the OnFiltering event handler. Be sure that the interactions between the Filter property and the OnFiltering event handler do not result in an empty filter set when they are used simultaneously in an application.

Event OnFiltering example

```
procedure TForm1.RxFilterOnFiltering(Sender: TObject; DataSet: TDataSet): Boolean;
begin
    if (AnsiUpperCase(DataSet.FieldByName('Cust_Name')) = 'Smith') or
        ((DataSet.FieldByName('Rate') > 0.5) and (DataSet.FieldByName('Rate') < 0.95)) then
        Result := True
    else
        Result := False;
end;
```

Event OnReleaseCapture

Declaration: OnReleaseCapture: TNotifyEvent;

The OnReleaseCapture event is activated when a data-aware controls, previously captured by the filter, is released by calling the ReleaseCapture method. By assigning a method to this property, you can take any special actions required by the event.

Event OnSetCapture

Declaration: OnSetCapture: TNotifyEvent;

The OnSetCapture event is activated when a filter captures the data-aware controls assosiated with filtered dataset, by calling the SetCapture method. By assigning a method to this property, you can take any special actions required by the event.

Method Activate

Declaration: procedure Activate;

Activate filter for a dataset. It is equivalent to setting the Active property to True.

Method Deactivate

Declaration: procedure Deactivate;

Deactivate filter for a dataset. It is equivalent to setting the Active property to False.

Method ReadCaptureControls

Declaration: procedure ReadCaptureControls;

You can use same data-aware controls (i.e. TDBEdit, TDBCheckBox etc) to display and edit dataset's records and to enter filter's selection criteria. Use SetCapture, ReleaseCapture and ReadCaptureControls methods to enable end-users to enter selection criteria into data-aware controls at run-time.

SetCapture method "captures" data-aware controls linked to a same datasource as TRxDBFilter component and allows user to enter data into these controls as selection criteria (without change the state of a dataset). You can read entered data by calling ReadCaptureControls method. This method reads selection criteria from data-aware controls captured by the filter component and fills the its Filter property by values readed. The single ctiteries combines into Filter property by using AND or OR logic operation according to LogicCond property value. After entering and reading selection criteria you MUST return data-aware controls to its normal state by calling ReleaseCapture method.

Method ReleaseCapture

Declaration: procedure ReleaseCapture;

You can use same data-aware controls (i.e. TDBEdit, TDBCheckBox etc) to display and edit dataset's records and to enter filter's selection criteria. Use SetCapture, ReleaseCapture and ReadCaptureControls methods to enable end-users to enter selection criteria into data-aware controls at run-time.

After using SetCapture method to entering selection criteria into data-ware controls and reading this criteria by ReadCaptureControls method you MUST return data-aware controls into its normal state by calling ReleaseCapture method.

Calling ReleaseCapture triggers the OnReleaseCapture event handler if one is defined for the TRxDBFilter component.

Method SetCapture

Declaration: procedure SetCapture;

You can use same data-aware controls (i.e. TDBEdit, TDBCheckBox etc) to display and edit dataset's records and to enter filter's selection criteria. Use SetCapture, ReleaseCapture and ReadCaptureControls methods to enable end-users to enter selection criteria into data-aware controls at run-time. SetCapture method "captures" data-aware controls linked to a same datasource as TRxDBFilter component and allows user to enter data into these controls as selection criteria (without change the state of a dataset). You can use entered data by calling ReadCaptureControls method. After entering and reading selection criteria you must return data-aware controls to its normal state by calling ReleaseCapture method. Calling SetCapture triggers the OnSetCapture event handler if one is defined for the TRxDBFilter component.

Method Update

Declaration: procedure Update;

The Update method reset the filter on the filtered dataset.

Method UpdateFuncFilter

Declaration: procedure UpdateFuncFilter;

Update resets the callback-filter on the filtered dataset. This method must be called when you change the conditions in OnFiltering event handler at runtime.

Type EFilterError

Declaration: EFilterError = class(Exception);

The EFilterError type is the exception type raised when an error is detected by the TRxDBFilter component.

Type TDBFilterOption

Declaration: TDBFilterOption = (foCaseInsensitive, foNoPartialCompare);

The TFilterOptions type defines the possible values for the Options property of the TRxDBFilter component.

Type TDBFilterOptions

Declaration: TDBFilterOptions = set of TDBFilterOption;

The TFilterOptions type defines the possible values for the Options property of the TRxDBFilter component.

Type TFilterEvent

Declaration: TFilterEvent = function (Sender: TObject; DataSet: TDataSet) : Boolean of object;

The TFilterEvent type points to a method that responds to an OnFiltering event of the TRxDBFilter component, for including or excluding records from being visible in a dataset.

Type TFilterLogicCond

Declaration: TFilterLogicCond = (flAnd, flOr);

The TFilterLogicCond type defines the possible values for the LogicCond property of the TRxDBFilter component.

Routine DropAllFilters

Declaration: procedure DropAllFilters(DataSet: TDataSet);

DropAllFilter drops the all filters on the specified DataSet and frees all resources associated with the filters.

DropAllFilters example: begin

... DropAllFilters(Query1); end;

Unit DbFilter

Event OnActivate OnDeactivate OnFiltering OnReleaseCapture OnSetCapture

Method Activate Deactivate ReadCaptureControls ReleaseCapture SetCapture UpdateFuncFilter

Property Active Captured ExprFilter Filter FuncFilter LogicCond Options Priority

Routine DropAllFilters

Type <u>EFilterError</u> <u>TDBFilterOption</u> <u>TFilterEvent</u> <u>TFilterLogicCond</u>

Control Name	Unit	Class
RX DB Grids	DbGrid	TRxDBGrid

Description: TRxDBGrid component provides you with the ability to:

- change the background color and font displayed within individual cells and entire rows and columns (with OnGetCellParams, OnGetBtnParams event);
- save and restore columns order and display width in ini-files or system registry (using the SaveLayout, RestoreLayout, SaveLayoutReg, RestoreLayoutReg methods or IniStorage property);
- display icons for BLOB, memo, OLE and picture fields (if ShowGlyphs property is True);
- select multiple records (if MultiSelect = True);
- convert columns headings to buttons (TitleButtons property, OnTitleBtnClick event);
- define fixed, non-scrollable columns in the left-hand side of the grid (using FixedCols property).

Using class TRxDBGrid

There is example of using TRxDBGrid to display customers data (CUSTOMER.DB table from DBDEMOS alias) with using OnGetCellParams event:

procedure TForm1.rxDBGrid1GetCellParams(Sender: TObject; Field: TField; AFont: TFont; var Background: TColor; Highlight: Boolean);

```
var
  OldDate, NewDate: TDateTime;
  DataSet: TDataSet;
begin
  if Field.FieldName = 'Company' then
    AFont.Style := AFont.Style + [fsBold]
  else
    if Field.FieldName = 'Contact' then
       if not Highlight then Background := clYellow:
  if not Highlight then begin
    OldDate := EncodeDate(1990, 1, 1);
    NewDate := EncodeDate(1995, 1, 1);
    DataSet := (Sender as TrxDBGrid).DataSource.DataSet;
  if DataSet.FieldByName('LastInvoiceDate').AsDateTime < OldDate then
    Background := clLime { oldest } else
  if DataSet.FieldByName('LastInvoiceDate').AsDateTime >= NewDate then
    AFont.Color := clRed; { newest }
  end;
end;
```

Use the TRxDBGrid component to display data from a dataset component in a spreadsheet-like grid instead standard TDBGrid component.

Property ClearSelection

Declaration: ClearSelection: Boolean;

Property FixedCols

Declaration: FixedCols: Integer;

The FixedCols property can be used to specify a number of fixed columns in the grid. The number of fixed columns must always be less than the total number of columns or fields in the grid. Fixed columns are always read-only when adding or editing records. This allows program or database generated fields such as record keys to be locked from user update. Values can be added to these fields by the DBMS when posting a record; in the AfterInsert event, in the OnNewRecord or in the BeforePost event of the linked dataset. The value can be set from 0 to one less than the total number of columns (or fields in the dataset).

Associated with the FixedCols property is a FixedColor property which allows you to change the color of the fixed columns.

Property InplaceEditor

Declaration: InplaceEditor: TInplaceEditor;

Runtime only. InplaceEditor is the default editor used by the grid to edit a cell's value. You may wish to reference this property if you need to access the currently edited value or dynamically manipulate the editor. For instance, if you wish to change the color of the editor when it has an invalid value.

Property MultiSelect

Declaration: MultiSelect: Boolean;

When True, the user can select multiple, non-contiguous rows using Ctrl+click or Shift+arrow keys. The behavior is similar to a multi-select list box. The records you select are represented as bookmarks and are stored in the SelectedRows property. The SelectedRows property is an object of type TBookmarkList.

Property SelectedRows

Declaration: SelectedRows: TBookmarkList;

SelectedRows is a set of bookmarks to all the records in the dataset that correspond to rows selected in the grid. Use the properties and methods of the TBookmarkList object returned by

SelectedRows to

- Determine the number of rows in the grid that are selected.
- Determine whether the current record in the dataset is selected.
- Determine whether a particular record in the dataset is selected.
- Delete all selected rows from the dataset.

SelectedRows is only meaningful when the MultiSelect property is True (or Options property includes dgMultiSelect in 32-bit version). You can navigate through selected rows in the dataset by using GotoSelection method of TRxDBGrid.

Property ShowGlyphs

Declaration: ShowGlyphs: Boolean;

This option would allow bitmaps or glyphys to be displayed.

Property TitleButtons

Declaration: TitleButtons: Boolean;

When this property is True, the column headings of each column will act as a button. When the user clicks on one, it will depress and fire the OnTitleBtnClick event. You can also handle OnCheckButton and OnGetBtnParams events to change title buttons property.

Event OnCheckButton

Declaration: OnCheckButton: TCheckTitleBtnEvent;

When the checkbutton is checked or unchecked, this event is triggered.

Event OnGetBtnParams

Declaration: OnGetBtnParams: TGetBtnParamsEvent;

This event which executes just prior to painting each title cell (when TitleButtons is True), allows you to change both the font and background colors of each title button.

Event OnGetCellParams

Declaration: OnGetCellParams: TGetCellParamsEvent;

This event, which executes just prior to painting the field values within each grid cell, allows you to change both the font and background colors of individual cells.

Event OnGetCellProps

Declaration: OnGetCellProps: TGetCellPropsEvent;

Obsolete event (for backward compatibility only). Use OnGetCellParams instead.

Event OnTitleBtnClick

Declaration: OnTitleBtnClick: TTitleClickEvent;

The OnTItleBtnClick event occurs when the user clicks on one of the column headings in the TRxDBGrid. The TitleButtons property must be True in order for this event to occur.

Method GotoSelection

Declaration: procedure GotoSelection(Index: Integer);

This method moves the index to that particular row or selection.

Method RestoreLayout

Declaration: procedure RestoreLayout(IniFile: TIniFile);

From the inifile, this method extracts the layout property information and restores the property information to the column, rows of the string-grid.

See also: <u>FormStorage</u>.

Method RestoreLayoutReg

Declaration: procedure RestoreLayoutReg(IniFile: TRegIniFile);

From the inifile, this method extracts the layout property information and restores the property information to the column, rows of the string-grid.

See also: <u>FormStorage</u>.

Method SaveLayout

Declaration: procedure SaveLayout(IniFile: TIniFile);

This method saves the current layout information of the DbGrid to the INI file.

Method SaveLayoutReg

Declaration: procedure SaveLayoutReg(IniFile: TRegIniFile);

This method saves the current layout information of the DbGrid to the INI file.

Method SelectAll

Declaration: procedure SelectAll;

This method adds all records to the list of currently selected records (if MultiSelect=True).

Method ToggleRowSelection

Declaration: procedure ToggleRowSelection;

This method adds or removes the record relating to the current grid row to the list of currently selected records if MultiSelect property is True. For instance, you can call this method from OnKeyPress event handler.

Method UnselectAll

Declaration: procedure UnselectAll;

This method removes all records from the list of currently selected records (if MultiSelect=True).

Type TBookmarkList

Declaration: TBookmarkList = class;

When you set MultiSelect property of a TRxDBGrid to True, you give yourself the ability to select multiple records within the grid. The records you select are represented as bookmarks and are stored in the SelectedRows property. The SelectedRows property is an object of type TBookmarkList. The properties and methods are described below.

The Items property is a List of TBookmark. The Count property returns the number of currently selected items in the DBGrid The Clear method will free all the selected records within the DBGrid

- The Delete method will delete all the selected rows from the dataset
- The Find method determines whether a bookmark is in the selected list.
- Use the Find method to locate the position of the bookmarked record within the selected list in the DBGrid.
- The IndexOf method returns the index of the bookmark within the Items property.
- The Refresh method returns a boolean value to notify whether any orphans were dropped (deleted) during the time the record has been selected in the grid.
- The Refresh method can be used to update the selected list to minimize the possibility of accessing a deleted record.
- The CurrentRowSelected property returns a boolean value and determines whether the current row is selected or not.

NOTE: In 32-bit version Delphi this class is declared in DBGRIDS.PAS unit.

Type TCheckTitleBtnEvent

Declaration: TCheckTitleBtnEvent = procedure (Sender: TObject; ACol: Longint; Field: TField; var Enabled: Boolean) of object;

TCheckTitleBtnEvent is the type of the OnCheckButton event of the TRxDBGrid component.

Type TDBLabelOptions

Declaration: TDBLabelOptions = (doCaption, doGlyph, doBoth);

TDBLabelOptions is a set of available values for the ShowOptions property of the TDBStatusLabel control.

Type TDBLabelStyle

Declaration: TDBLabelStyle = (lsState, lsRecordNo);

TDBLabelStyle is a set of available values for the Style property of the TDBStatusLabel control.

Type TGetBtnParamsEvent

Declaration: TGetBtnParamsEvent = procedure (Sender: TObject; Field: TField; AFont: TFont; var Background: TColor; IsDown: Boolean) of object;

TGetBtnParamsEvent is the type of the OnGetBtnParams event of the TRxDBGrid component.

Type TGetCellParamsEvent

Declaration: TGetCellParamsEvent = procedure (Sender: TObject; Field: TField; AFont: TFont; var Background: TColor; Highlight: Boolean) of object;

TGetCellParamsEvent is the type of the OnGetCellParams event of the TRxDBGrid component.

Type TGetCellPropsEvent

Declaration: TGetCellPropsEvent = procedure (Sender: TObject; Field: TField; AFont: TFont; var Background: TColor) of object;

TGetCellPropsEvent is the type of the OnGetCellProps (obsolete) event of the TRxDBGrid component.

Type TGetStringEvent

Declaration: TGetStringEvent = function(Sender: TObject): string of object;

The TGetStringEvent points to a method that handles the retrieving of the string displayed as a dataset name in a TDBStatusLabel.

Type TGlyphAlign

Declaration: TGlyphAlign = glGlyphLeft..glGlyphRight;

TGlyphAlign is a set of available values for the GlyphAlign property of the TDBStatusLabel control.

Type TTitleClickEvent

Declaration: TTitleClickEvent = procedure (Sender: TObject; ACol: Longint; Field: TField) of object;

TTitleClickEvent is the type of the OnTitleClick event of the TRxDBGrid component.

Unit DbGrid

Using class TRxDBGrid Event OnCheckButton OnGetBtnParams OnGetCellParams OnTitleBtnClick

Method GotoSelection RestoreLayout SaveLayout SaveLayoutReg SelectAll ToggleRowSelection UnselectAll

Property ClearSelection FixedCols InplaceEditor MultiSelect SelectedRows ShowGlyphs TitleButtons

Type TBookmarkList TCheckTitleBtnEvent TDBLabelOptions TDBLabelStyle TGetBtnParamsEvent TGetCellParamsEvent TGetCellPropsEvent TGetStringEvent TGlyphAlign TTitleClickEvent

Control Name	Unit	Class
RX Date DB Edit	RxDBCtrl	TDBDateEdit

Description:

TDBDateEdit is a direct descendant of the TDateEdit and inherits all of its properties and methods. There are very few differences in the behavior of the TDBDateEdit and TDateEdit except that TDBDateEdit connects to a DataSource and allows visual manipulation of a Date or DateTime field.

TDBDateEdit provides three additional properties. The DataField, DataSource, and Field properties are exactly the same as the like-named properties in Delphi's standard data-aware components.

Using a TDBDateEdit component is very simple - assign a TDataSource to the DataSource property and assign a date or date-time field name to the DataField property. The TDBDateEdit control automatically gets the date from the data field and updates the data source with the changed data.

To control 2- or 4-digit year in TDateEdit, TDBDateEdit components you can use a typed boolean constant FourDigitYear from DateUtil unit.

Property BlanksChar

Declaration: BlanksChar: Char;

BlanksChar is the character used to represent unentered characters in the date mask (default value is space character - #32).

Property CalendarHints

Declaration: CalendarHints: TStrings;

CalendarHints provides a way to customize the Help Hints for the buttons on popup-calendar or calendar dialog.

Each button on calendar (prev year, prev month, next month, next year) has a default Help Hint.

CalendarHints allow the values of any or all of these default Help Hints to be replaced by customized hints.

CalendarHints is a string list. Each hint is a string. The first string in the string list becomes the Help Hint for the first button on the calendar (the "Prev Year" button). The 4-th hint becomes the Help Hint for the last 4-th button (the "Next Year" button).

When specifying CalendarHints at runtime, enter an empty string (") for any Help Hint that should keep the default value. Simply leave the line blank when using the string list property editor of the Object Inspector for the CalendarHints property.

Property CalendarStyle

Declaration: CalendarStyle: TCalendarStyle;

The CalendarStyle property sets the different types of popup calendars. If the style is set to csDialog, then the popup calendar will take the form of a Dialog Box with captions and borders. The dialog's caption can be set by the DialogTitle property. If the style is set to csPopup (by default), then the calendar will appear as if it were a combo box. It will contain no title or borders. It will simply be the calendar. The PopupCalendarSize global variable from PickDate unit can be used to control the size of pop-up calendar.

Property CheckOnExit

Declaration: CheckOnExit: Boolean;

When True, if an invalid date value is entered in the TDateEdit or TDBDateEdit by the user, the exception will be raised when the user exits the date-editor.

Property Date

Declaration: Date: TDateTime;

This property (run-time only) determines the date that the TDateEdit displays. Assigning a value to Date causes the TDateEdit to display that date as its active date.

Property DefaultToday

Declaration: DefaultToday: Boolean;

If the DefaultToday property is True and the Date value is null (edit text is empty) or invalid then the current date is returneded by the Date property.

Property DialogTitle

Declaration: DialogTitle: string;

The DialogTitle property determines the text that appears in the pop-up dialog box's title bar.

Property PopupColor

Declaration: PopupColor: TColor;

The PopupColor is the background color of the pop-up calendar.

Property PopupVisible

Declaration: PopupVisible: Boolean;

PopupVisible (read-only and runtime-only property) specifies whether the popup-calendar is open or "dropped-down" (when CalendarStyle is csPopup). Read PopupVisible to determine whether the popup calendar is currently in the open (dropped-down) position.

Property StartOfWeek

Declaration: StartOfWeek: TDayOfWeekName;

StartOfWeek determines the day of the week that begins a week. This property determines which day of the week is drawn in the left column of the pop-up calendar.

Property WeekendColor

Declaration: WeekendColor: TColor;

WeekendColor determines the colors used for the pop-up calendar. It is the color set for the "free" days (weekends) in the month.

_

Property Weekends

Declaration: Weekends: TDaysOfWeek;

WeekEnds allows the programmer defines which days are weekends.

UNIT PickDate

Const <u>PopupCalendarSize</u>

Routine

PopupDate SelectDate SelectDateStr

Const PopupCalendarSize

Declaration: PopupCalendarSize: TPoint = (X: 187; Y: 124);;

The PopupCalendarSize global variable can be used to control the size of pop-up calendar in TDateEdit and TDBDateEdit controls.

Routine PopupDate

Declaration: function PopupDate(var Date: TDateTime; Edit: TWinControl): Boolean;

PopDate function pops-up the calendar with Date on the control *Edit*.

PopupDate example:

if PopupDate(Date1, Edit1) then
begin
 {...}
end;

Routine SelectDate

Declaration: function SelectDate(var Date: TDateTime; const DlgCaption: TCaption; AStartOfWeek: TDayNameOfWeek; AWeekends: TDaysOfWeek; AWeekendColor: TColor; BtnHints: TStrings): Boolean;

function SelectDate pops up a dialog box with *Date* as a variable to accept the date. The *DlgCaption* is the caption of the dialogbox. *AStartOfWeek* defines the start of the week, while AWeekEnds defines the dates

which are the weekends, and AWeekEndColour is the AWeekEnds label'S colour and *BtnHints* are the hints strings.

SelectDate example: D := SysUtils.Date; { set D to current date } if SelectDate(D, 'Select a date', Mon, [Sun], clRed, nil) then begin { use selected date } end;

Routine SelectDateStr

Declaration: function SelectDateStr(var StrDate: string; const DlgCaption: TCaption; AStartOfWeek: TDayNameOfWeek; AWeekends: TDaysOfWeek; AWeekendColor: TColor; BtnHints: TStrings): Boolean;

SelectDateStr is the same as <u>selectdate</u> with the variable *Date* returned as a string is.

SelectDateStr example:

SD := DateToStr(SysUtils.Date); { set SD to current date }
if SelectDateStr(SD, 'Select a date', Mon, [Sun], clRed, nil) then
begin
 { use selected date }
end;
Class <u>TRxDBCalcEdit</u>

Property BlanksChar CalendarHints CalendarStyle CheckOnExit Date DefaultToday DefaultToday DialogTitle PopupColor PopupVisible StartOfWeek WeekendColor Weekends

Control Name	Unit	Class
RX Database Labels	<u>dbcombo</u>	TDBStatusLabel

Description:

TDBStatusLabel is a data-aware label used to show state information or position information about DataSources. It can be used on any dataset to report the current state, i.e. Inactive, Browse, Edit, Insert, Set Key, Calculating Fields; or on Paradox and dBase databases or on memory tables (TMemoryTable component) to show logical or physical record position and number of records.

When the Style property is lsState (by default) the <dataset_name>: <dataset_state> mnemonic is used to place the state label for the current state.

When the Style property is lsRecordNo the <current_record_number>:<full_record_count> mnemonic is used to place the record number and the total number of records within the string. TDBStatusLabel requires the DataSource property to be set to bind it to a dataset.

Property Captions

Declaration: Captions: TStrings;

Property DataSetName

Declaration: DataSetName: string;

Name contains the name of the dataset as referenced by other components.

Property EditColor

Declaration: EditColor: TColor;

When the database is in edit mode, TDBStatusLabel changes to the colour specified by EditColor.

Property GlyphAlign

Declaration: GlyphAlign: TGlyphAlign;

This property determines the location of a glyph displayed within a status label. The glyph can be placed either to the left (the default) or to the right of the caption.

Property Layout

Declaration: Layout: TTextLayout;

Layout specifies the vertical placement of the text within the label when the AutoSize property is False. Set Layout to specify how the text of the label is placed within the ClientRect of the label control .

Layout can be one of the following values:

- tlTop The text appears at the top of the label.
- tlCenter The text is vertically centered in the label.
- tlBottom The text appears along the bottom of the label.

Property ShowOptions

Declaration: ShowOptions: TDBLabelOptions;

Use this property to hide and show the caption and glyph of a TDBStatusLabel component.

Property Style

Declaration: Style: TDBLabelStyle;

Style defines the style defined by:

- IsState TDBStatusLabel displays the state of the table.
- IsRecNo TDBStatusLabel displays the current record number.

Event OnGetDataName

Declaration: OnGetDataName: TGetStringEvent;

OnGetDateName event is triggered when the OnGetDataName is called.

Class <u>RX Db Combo</u>

Property EnableValues Values

Class <u>RX DB Lookup Lists</u>

Property EmptyItemColor IndexSwitch ListStyle LookupDisplayIndex RowCount Value

Event OnGetImage

Type <u>TGetImageEvent</u> <u>TLookupListStyle</u>

Class <u>RX Database Labels</u>

Property Captions DataSetName EditColor GlyphAlign Layout ShowOptions SQLCount Style

Events OnGetDataName

Control Name	Unit	Class
TRxDBLookupList	<u>RxLookup</u>	TRxDBLookupList

Description:

TRxDBLookupList provides a list of lookup items for filling in fields that require data from another dataset. Use TRxDBLookupList to provide users with a convenient list of lookup items to set a field value using the values of a field in another dataset. Lookup list boxes usually display values that are a represent a more meaningful description of the actual field value. The relationship between field values and the corresponding values in the lookup dataset can be set using the properties LookupSource, LookupField and LookupDisplay.

This component provides the following:

- You can select any number of fields to be displayed in the lookup list.
- End-users can incrementally search through the lookup list by directly typing into the control. This is a great advantage when using lookup tables that contain hundreds of even thousands of records. You can perform a lookup on a Query or QBE result. It even incrementally searches on the query result.
- The component does not have to be bound, or assigned, to a table's field (DataField and DataSource properties) which gives you greater flexibility in using this lookup list for general tasks where a source table is not involved.

If DataSource and DataField properties is set, when a user selects a list item, the corresponding field value is changed in the underlying dataset.

If IgnoreCase property is True (default), the incrementally search through the lookup list is processed without regard to case in the dataset's data.

If <u>IndexSwitch property</u> is True (default) and TTable component is linked to the LookupSource, the incrementally search will use available indices of lookup table.

The DisplayEmpty property allows you specify the text value to display when no item is selected from the lookup list. If DisplayEmpty set to the non-empty string, this value will be show in the top of the list additionaly to the lookup items.

You can also use an OnGetImage event to specify graphical picture to display in each item of lookup list accordingly to the contents of lookup source.

Property EmptyItemColor

Declaration: EmptyItemColor: TColor;

The EmptyItemColor property allows you specify the color value to display additional non-scrolled "empty" item in the top of the list if DisplayEmpty set to the non-empty string.

Property IndexSwitch

Declaration: IndexSwitch: Boolean;

If IndexSwitch property is True (default) and TTable component is linked to the LookupSource, the incrementally search will use available indices of lookup table.

Property ListStyle

Declaration: ListStyle: TLookupListStyle;

ListStyle determines how the lookup list displays its items when multiple fields specified by the LookupDisplay property. By default, style is lsFixed, meaning that the each display field always takes up a fixed width accordingly to their DisplayWidth properties. The lsDelimited style meaning that each field takes a variable width, and the fields are separated by a comma (',') character.

Property LookupDisplayIndex

Declaration: LookupDisplayIndex: Integer;

When multiple fields specified by the LookupDisplay property, the LookupDisplayIndex specifies index of a field in the LookupDisplay list which will be use in the DisplayValue property and will be display in the editor of the TRxDBLookupCombo component.

Property RowCount

Declaration: RowCount: Integer;

Set RowCount to the number of rows displayed in the lookup list box. The list box is resized to match the specified number of rows. Resizing the list box (such as by changing its Height property) automatically changes the value of RowCount to match the new height.

Property Value

Declaration: Value: string;

Run-time only. The value of the Value property is the contents of the DataField for the current record in the primary dataset. As the user moves through the primary dataset, the value of the Value property changes.

By explicitly changing the Value property value at run time, you change the contents of the field.

Event OnGetImage

Declaration: OnGetImage: TGetImageEvent;

OnGetImage event is triggered when an image is requested.

Type TGetImageEvent

Declaration: TGetImageEvent = procedure (Sender: TObject; IsEmpty: Boolean; var Graphic: TGraphic;

var TextMargin: Integer) of object;

TGetImageEvent is the type of the OnGetImage event of the TRxDBLookupList and the TRxDBLookupCombo components.

Type TLookupListStyle

Declaration: TLookupListStyle = (lsFixed, lsDelimited);

The TLookupListStyle type is the type of the ListStyle property for a lookup list controls (TRxDBLookupList and TRxDBLookupCombo components).

Control Name	Unit	Class
TRxDBLookupCombo	<u>RxLookup</u>	TRxDBLookupCombo

Description:

The TRxDBLookupCombo visual interface component provides your end-users with the ability to select a value for a field from a drop-down list of values that is populated from a second lookup table.

If TRxDBLookupCombo is linked to a lookup field component, it automatically reads the relationship between the field value and the lookup values in the lookup dataset from the field component. The relationship between field values and the corresponding values in the lookup dataset can also be explicitly set using the properties of the lookup combo box when the combo box is not linked toa lookup field component.

This component provides the following:

- You can select any number of fields to be displayed in the drop-down list.
- End-users can incrementally search through the lookup list by directly typing into the combo control while the lookup list is displayed. This is a great advantage when using lookup tables that contain hundreds of even thousands of records. You can perform a lookup on a Query or QBE result. It even incrementally searches on the query result.
- The component does not have to be bound, or assigned, to a table's field (DataField and DataSource properties) which gives you greater flexibility in using this LookupCombo for general tasks where a source table is not involved.

If DataSource and DataField properties is set, when a user selects a list item, the corresponding field value is changed in the underlying dataset.

If IgnoreCase property is True (default), the incrementally search through the lookup list is processed without regard to case in the dataset's data.

If IndexSwitch property is True (default) and TTable component is linked to the LookupSource, the incrementally search will use available indices of lookup table.

The <u>DisplayEmpty</u> property allows you specify the text value to display when no item is selected from the lookup list. If DisplayEmpty set to the non-empty string, this value will be show in the drop-down list additionaly to the lookup items.

Use DropDownAlign property to specify how the drop-down list is aligned relative to its edit box.

Use an OnGetImage event to specify graphical picture to display in each item of lookup list accordingly to the contents of lookup source.

Write an OnChange event handler to take specific action immediately after the user selects an item from the list and the <u>Value</u> property changed.

Property DisplayEmpty

Declaration: DisplayEmpty: string;

The DisplayEmpty property allows you specify the text value to display when no item is selected from the lookup list and the Value property is equal to the EmptyValue property (empty string by default). If DisplayEmpty set to the non-empty string, this value will be show at the top of the list additionally to the lookup items. The EmptyItemColor value use to display background of this item.

Property EmptyValue

Declaration: EmptyValue: string;

If database is empty, the variable EmptyValue replaces the table.

Property EscapeClear

Declaration: EscapeClear: Boolean;

When the DisplayEmpty property is set to empty string, the user is not able to clear their selection. The EscapeClear property when set to True (by default), gives the user a convenient way to clear the combos current selection simply by entering the <ESCAPE> character.

Property IgnoreCase

Declaration: IgnoreCase: Boolean;

If IgnoreCase property is True (default), the incrementally search through the lookup list is processed without regard to case in the dataset's data. When False, incremental searching considers case sensitivity when performing an incremental search.

This property has no effect with some SQL-Links.

Property KeyValue

Declaration: KeyValue: Variant;

Use KeyField to determine the value represented by the lookup control (not the value displayed by the lookup control). When KeyValue is set, the lookup control attempts to find a record from the LookupSource's dataset where the value of LookupField matches KeyValue. If such a match is found, the lookup control displays the value of LookupDisplay on that record.

Property ListVisible

Declaration: ListVisible: Boolean;

ListVisible specifies whether the lookup list is open or "dropped-down". Read ListVisible to determine whether the list of lookup values is currently in the open (dropped-down) position. If ListVisible is True, the list is open; if ListVisible is False, the list is closed.

Property Value

Declaration: Value: string;

Run-time only. The value of the Value property is the contents of the DataField for the current record in the primary dataset. As the user moves through the primary dataset, the value of the Value property changes.

By explicitly changing the Value property value at run time, you change the contents of the field.

Event OnCloseUp

Declaration: OnCloseUp: TNotifyEvent;

OnCloseUp occurs immediately after an opened or "dropped-down" list is closed. Write an OnCloseUp event handler to respond when the combo box list is closed. When the list is closed, the value that corresponds to the selected lookup value is assigned to the field. The list can be closed by the user or by calling the CloseUp method.

Method CloseUp

Declaration: procedure CloseUp(Accept: Boolean);

The CloseUp method closes an opened or "dropped-down" database lookup combo box. The Accept parameter determines whether to modify the DataField with the selected value in the combo box.

Method IsDropDown

Declaration: function IsDropDown: Boolean;

IsDropDown method returns True if the lookup list is open or "dropped-down"; if IsDropDown returns False, the list is closed.

Method ResetField

Declaration: procedure ResetField;

ResetField method sets the Value property to the EmptyValue and the DisplayValue to the DisplayEmpty.

Property ClickKey

Declaration: ClickKey: TShortCut;

The ClickKey property determines the key strokes users can use to press a button by keyboard. Default value is "Alt+Down".

Property DirectInput

Declaration: DirectInput: Boolean;

Set this property to True to allow a user to type in a string value into the edit portion. When this property is False, only the button and OnButtonClick can be used to modify the text in the editor.

Unit <u>RX lookup</u>

Class <u>RX Database Lookup Combo</u> Property <u>DisplayEmpty</u> <u>EmptyValue</u> <u>EscapeClear</u> <u>IgnoreCase</u> KeyValue <u>ListVisible</u> Value

Events <u>OnCloseUp</u>

Method <u>CloseUp</u> <u>IsDropDown</u> <u>ResetField</u>

Property <u>ClickKey</u> <u>DirectInput</u>

Control Name	Unit	Class
TRxDBComboBox	RxDBComb	TRxDBComboBox

Description:

TRxDBComboBox expands the capabilities of a regular data aware combo-box. It allows you to enter mapped storage and display values so that you can display understandable text versions of stored codes in your table, instead of displaying only the codes themselves where users have to remember what they all mean. Alternatively you could use a TRxDBLookupCombo to display one field from a lookup source, and store a different field, but the TRxDBComboBox's drop-down list comes directly from a string list and not required the lookup source.

Property EnableValues

Declaration: EnableValues: Boolean;

Property EnableValues forces the values found in <u>Values</u> to be displayed instead.

Property Values Declaration: Values: TStrings;

Property Values stores selections without connecting the second table to a database (e.g., male, female).

Control Name	Unit	Class
RX Database Rich Edit	DBRichEd	TDBRichEdit

Description: A TDBRichEdit object is a multiline edit control that can display and edit a rich text memo field in a dataset.

Use a TDBRichEdit object to enable users to edit a database field in a rich text edit control.

TDBRichEdit provides the properties and methods to enter and work with rich text. TRichEdit does not provide any user interface components to make these formatting options available to the user.

Applications must implement the user interface components to surface the rich text capabilities of a TDBRichEdit object.

TDBRichEdit uses the Text property to represent the contents of the field. If the database field does not include text formatting information, consider using a TDBMemo object or a TDBEdit object instead. If the application doesn't require the data-aware capabilities of TDBRichEdit, use a rich text edit control (TRichEdit) instead, to conserve system resources.

Control Name	Unit	Class
RX Database Index Combo	DBIndex	TDBIndexCombo

Description:

TDBIndexCombo is a descendant of the TCustomComboBox component and provides a drop-down list of the indexes available in the attached dataset. The list of indexes can be configured to display the index name in several ways to include a user-defined string for each index.

Selecting one of the index names from the drop-down list changes the dataset's active index so that future operations on that dataset will use the newly selected index.

Property DispalyMode

Declaration: DispalyMode: TIdxDisplayMode;

DisplayMode determines the text that is displayed in the drop-down list. The following table list possible values for DisplayMode and their meanings:

- dmFieldLabels displays the DisplayLabel of the fields in the index;
- dmFieldNames displays the list of field names that make up the index;
- dmIndexName displays the actual index name.

Property EnableNoIndex

Declaration: EnableNoIndex: Boolean;

EnableNoIndex property determines whether or not the popup list contains additional item which can be selected by end-user to defeat using any indexes (i.e. assign empty value to IndexName and IndexFieldNames properties of attached table). String value for this additional item specified by NoIndexItem property.

Property NoIndexItem

Declaration: NoIndexItem: string;

NoIndexItem property specifies the text string for additional item which can be selected by end-user to defeat using any indexes (i.e. assign empty value to IndexName and IndexFieldNames properties of attached table). This additional item used by TDBIndexCombo control only if EnableNoIndex property is True.

Type TIdxDisplayMode

Declaration: TIdxDisplayMode = (dmFieldLabels, dmFieldNames, dmIndexName);

The TIdxDisplayMode type defines the possible values for the DisplayMode property of the TDBIndexCombo component.

UNIT **DbIndex**

Property DispalyMode EnableNoIndex NoIndexItem

Type <u>TIdxDisplayMode</u>

Control Name	Unit	Class
RX Database Progress	DBPrgrss	TDBProgress

Description: TDBProgress is a wrapper for a Borland Database Engine (BDE) generic progress callback and (in 32-bit version only) trace callback functions. Generic progress callback is issued by BDE to inform applications about the progress made during large batch operations, such as DbiBatchMove. The Generic Progress Report callback allows the client to obtain progress reports during an operation, and to cancel the operation, if desired. This callback is generated when working with Paradox driver and with some other BDE drivers. The query execution generates some text messages and BatchMove execution generates text and percent notifications. You can specify the MessageControl property and Gauge property to display BDE messages and percentage.

For some operations, a percentage completed is passed back to the callback function, for most others a string containing progress information is passed back; for simple database operations no progress information is returned at all. It has been our experience that percentages are mainly passed back in operations where the BDE must physically sort a local database file, such as in indexing DBase files. It is of course possible to interpret the messages passed back and calculate your own percentage completed.

32-bit trace callback is a system-level callback that can be used to retrieve trace information.

Property TraceFlags specifies the database operations to track with the OnTrace event at run time while Trace property is True.

Using class TDBProgress

The TDBProgress component is a non-visual component that holds the latest progress information on the current database operation as returned by the database engine. By updating visual components, such as a gauge or a label, whenever the OnProgress, OnMessageChange and OnPercentChange events is generated, the user is aware of the status of the database action being executed. Such feedback is especially useful when performing database operations that consume a considerable amount of time, such as:

- Index Creation
- Query Execution
- BatchMoves
- etc.

The TDBProgress component enables you to see the actual statement calls made through SQL Links to a remote server or through the ODBC socket to an ODBC data source, like SQL Monitor. You can elect to monitor different types of activity. Choose TraceFlags property value to select different categories of activities to monitor.

Use the TDBProgress component to provide progress feedback on database operations and see the actual statement calls made through SQL Links to a remote server.

Property Active Declaration: Active: Boolean; Set the Active property to True to enables the progress callback-events. Depending on the value of the Active the DBProgress will generate the OnProgress, OnMessageChange and OnPercentChange events each time the database passes control back to the TDBProgress component during the current database operation. Setting this property to False disables the generic progress callbacks.

Property Gauge

Declaration: Gauge: TControl;

Gauge is a progress-control associated with the DBProgress to to display BDE percentage. In 16-bit Delphi version you can use TGauge control as progress-indicator. In 32-bit versions you can also use TProgressBar control.

Property MessageControl

Declaration: MessageControl: TControl;

MessageControl is a control associated with the DBProgress to to display BDE messages. You can use any control which displays it's Caption property, i.e. TLabel, TPanel etc.

Property Trace

Declaration: Trace: Boolean;

Set the Trace property to True to enables the trace callback-events. Setting this property to True enables the trace callback and depending on the value of the TraceFlags the DBProgress will generate the OnTrace events. Setting this property to False disables the trace callbacks.

Property WaitCursor

Declaration: WaitCursor: TCursor;

Indicates the mouse cursor changes to during long BDE operations.

Event OnMessageChange

Declaration: OnMessageChange: TOnMessageChange;

The OnPercentChange event occurs every time new text message is returned from the database engine (BDE). This event is only generated when Active is set to True.

Event OnPercentChange

Declaration: OnPercentChange: TOnPercentChange;

The OnPercentChange event occurs every time percent notifications is returned from the database engine (BDE). This event is only generated when Active is set to True.

Event OnProgress

Declaration: OnProgress: TOnProgressEvent;

The OnProgress event occurs every time progress information is returned from the database engine (BDE).

This event is only generated when Active is set to True. To abort the current database operation, set the Abort variable to False. The effect the abort will have depends of the nature of the database operation that is being aborted.

Event OnTrace

Declaration: OnTrace: TOnTraceEvent;

The OnTrace event occurs every time trace information (text string passed as Msg parameter) is returned from the database engine (BDE). This event is only generated when Trace property is set to True and TraceFlags property is not equal to empty set [].

Method ProgressMsgValue

Declaration: function ProgressMsgValue(const Msg: string): Longint;

Returns numeric value contained in BDE callback string specified by Msg parameter. The message string format always is: <Text_string><:><Value>. In the message string, the value and colon fields are optional. When message Msg contains value field, then ProgressMsgValue returns value as result, otherwise this method returns -1. You can use ProgressMsgValue method, for example, in OnMessageChange event handler.

Type TOnMessageChange

Declaration: TOnMessageChange = procedure(Sender: TObject; const Msg: string) of object;

The TOnMessageChange type points to the method that handles the OnMessageChange event of the TDBProgress component. The Msg parameter contains the last progress message passed back from the database engine.

Type TOnPercentChange

Declaration: TOnPercentChange = procedure(Sender: TObject; PercentDone: Integer) of object;

The TOnPercentChange type points to the method that handles the OnPercentChange event of the TDBProgress component. The PercentDone parameter contains the last progress percentage passed back from the database engine.

Type TOnProgressEvent

Declaration: TOnProgressEvent = procedure(Sender: TObject; var Abort: Boolean) of object;

TOnProgressEvent is the type of the OnProgress event of the TDBProgress component.

Type TOnTraceEvent

Declaration: TOnTraceEvent = procedure(Sender: TObject; Flag: TTraceFlag; const Msg: string) of object;

- 1

The TOnTraceEvent type points to the method that handles the OnTrace event of the TDBProgress component.

Unit DbPrgrs

Using class TDBProgress

Property Active Gauge MessageControl Trace WaitCursor

Event

OnMessageChange OnPercentChange OnProgress OnTrace

Method ProgressMsgValue

Туре

TOnMessageChange TOnPercentChange TOnProgressEvent TOnTraceEvent

Control Name	Unit	Class
RX Database Security	DBSecur	TDBSecurity

Description:

The TDBSecurity component provides a quick, standard interface for users to gain access to an databaseoriented application. This component provides two standard dialogs: registration (login) dialog and change user password dialog. Specify Database property or call Login method manually to show registration dialog, and call ChangePassword method to show change password dialog.

Also TDBSecurity allows you lock application in minimized state by calling Lock method. After lock user must enter password to unlock and restore application.

Using class TDBSecurity

Set Database, UsersTableName and LoginNameField to approperiate values, write event handler to the OnCheckUser event (if needed) and set Active to True for use standard login dialog when application started.

When UsersTableName and LoginNameField are not specified, you can use this component to connect to the SQL-oriented database.

TDBSecurity provides three standard dialogs for database-oriented applications:

- registration (login) dialog;
- change user password dialog;
- unlock with password dialog for unlock locked application.

Property Active

Declaration: Active: Boolean;

If there any IO access to the data, this is triggered

Property AllowEmptyPassword

Declaration: AllowEmptyPassword: Boolean; AllowEmptyPassword determines whether or not user can leave password edit box empty in the login dialog and in the change password dialog.

Property AttemptNumber

Declaration: AttemptNumber: Integer;

AttemptNumber is a count of available attempts to input valid user name and password in the login dialog. After user input invalid user name or password more then AttemptNumber times the application halts.

Property Database

Declaration: Database: TDatabase;

Specifies the Database component for which TDBSecurity component provides standard dialogs and login to by calling Login method.

Property LoggedUser

Declaration: LoggedUser: string;

Run-time and read-only property. Returns the user name logged to the database. You can use this property after successfull connection with database or in the handler of OnCheckUser event.

When connection to the database is unsuccess, the LoggedUser property is empty.

Property LoginNameField

Declaration: LoginNameField: string;

The LoginNameField property is a string which represents the name of a field in the table, specified by UsersTableName property. The field in question contains the login name of the user, and in most cases must be the first field in the existing index for the table.

Property MaxPasswordLen

Declaration: MaxPasswordLen: Integer;

The MaxPasswordLen property specifies the maximum number of characters the user can enter in an password edit box in the login dialog and in the change password dialog. The default setting for MaxPasswordLen is 0, which means that there is no limit on the number of characters the password can contain.

Property UpdateCaption

Declaration: UpdateCaption: TUpdateCaption;

UpdateCaption is used to update the Application's title, or form-caption if necessary, to inform the user of something important.

Property UseRegistry

Declaration: UseRegistry: Boolean;

UseRegistry determines whether or not a Windows System Registry will be used (in 32-bit version only) to store data. When UseRegistry is False, data will be store in text INI-file.

Property UsersTableName

Declaration: UsersTableName: TFileName;

The UsersTableName property points to an existing database table, which holds data about the users of the

application and their passwords. At the least, the table must have a field representing a login name, specified by LoginNameField property, and optionally a fields representing a password or any extraneous information, which can be used in the OnCheckUser event handler.

Event AfterLogin

Declaration: AfterLogin: TNotifyEvent;

After the user login, AfterLogin event is triggered.

Event OnChangePassword

Declaration: OnChangePassword: TChangePasswordEvent;

When the user changes the password, OngChangePassword is triggered.

Event OnChangePassword example:

```
function TMainForm.SecurityChangePassword(UsersTable: TTable; const OldPassword, NewPassword:
String): Boolean;
begin
    Result := False;
    if SecurityCheckUser(UsersTable, OldPassword) then
    begin
      with UsersTable do
      begin
      Edit;
      FieldByName('PASSWORD').AsString := CryptString(PswdKey, NewPassword);
      Post;
      Result := True;
      end;
      end;
    end;
```

Event OnCheckUser

Declaration: OnCheckUser: TCheckUserEvent;

Password checking

Event OnCheckUser example:

```
function TMainForm.SecurityCheckUser(UsersTable: TTable; const Password: String): Boolean;
begin
    Result := False;
    with UsersTable do
    begin
        if CryptString(PswdKey, FieldByName('PASSWORD').AsString) = Password then
        begin
```

```
Result := True;
UserId := FieldByName('ID').AsInteger;
UserLevel := FieldByName('USER_LEVEL').AsInteger;
end;
end;
end;
```

Event OnUnlock

Declaration: OnUnlock: TCheckUnlockEvent;

Event OnUnLock is triggered when the table is unlocked.

Method ChangePassword

Declaration: function ChangePassword: Boolean;

Uses the change-password instead.

Method Lock

Declaration: procedure Lock;

procedure locks locks the database.

Method Login

Declaration: function Login: Boolean;

The Login method displays the registration login dialog represented by the component on the screen, try connect to database and check user rights with event OnCheckUser. It's called automatically when Active property is True, and can be called manually. It returns true if the user succesfully logged into database.

Type TChangePasswordEvent

Declaration: TChangePasswordEvent = function(UsersTable: TTable; const OldPassword, NewPassword: string): Boolean of object;

TChangePasswordEvent is the type of the OnChangePassword event of the TDBSecurity component.

Type TCheckUnlockEvent

Declaration: TCheckUnlockEvent = function(const Password: string): Boolean of object;

TCheckUnlockEvent is the type of the OnUnlock event of the TDBSecurity component.

Type TCheckUserEvent

Declaration: TCheckUserEvent = function(UsersTable: TTable; const Password: string): Boolean of object;

-

TCheckUserEvent is the type of the OnCheckUser event of the TDBSecurity component.

Type TUpdateCaption

Declaration: TUpdateCaption = (ucNoChange, ucAppTitle, ucFormCaption);

The TUpdateCaption is the type for the UpdateCaption property of TDBSecurity component.

UNIT <u>DBSecur</u> <u>Using class TDBSecurity</u>

Property Active AllowEmptyPassword AttemptNumber Database LoggedUser LoginNameField MaxPasswordLen UpdateCaption UseRegistry UsersTableName

Event <u>AfterLogin</u> <u>OnChangePassword</u> <u>OnCheckUser</u> <u>OnUnlock</u> <u>ChangePassword</u> <u>Lock</u> <u>Login</u>

Type TChangePasswordEvent TCheckUnlockEvent TCheckUserEvent TUpdateCaption

Control Name	Unit	Class
RX BDE Items	DBExcpt	TRxBdeErrorDlg

Description:

The class TRxBdeErrorDlg replaces the normal message box for EDBEngineError exceptions raised at runtime.

The standard message box displayed by an unhandled exception shows only the exception's name and description. EDBEngineError exceptions, however, contain additional information that can be useful to application developers. For this reason, exceptions of the EDBEngineError class, appear in a special dialog box called TRxBdeErrorDlg. The dialog shows the entire stack of BDE errors as represented in the EDBEngineError object.

You can use the DbErrorIntercept procedure to display TRxBdeErrorDlg instead standard message box. This can be useful for debugging or in cases where you want the end user to have access to BDE error information.

You can also specify handler of <u>OnErrorMsg</u> event to change text of error message displayed by TRxBdeErrorDlg, and help context for error dialog by using <u>DbErrorHelpCtx</u> variable.

Event OnErrorMsg

Declaration: OnErrorMsg: TDBErrorEvent;

Use OnErrorMsg to perform special processing when the error message is shown. You can, for example, change the text of error message by using Msg parameter.

Const DbErrorHelpCtx

Declaration: DbErrorHelpCtx: THelpContext = 0;

The DbErrorHelpCtx variable is an integer value that determines which Help screen appears when the user requests context-sensitive online Help from the TRxBdeErrorDlg dialog by pressing "F1" key..

Type TDBErrorEvent

Declaration: TDBErrorEvent = procedure (Error: TDBError; var Msg: string) of object;

TDBErrorEvent is the type of the OnErrorMsg event of the TRxBdeErrorDlg dialog control.

Routine DbErrorIntercept

Declaration: procedure DbErrorIntercept;

DbErrorIntercept enables the TRxBdeErrorDlg instance, replacing the standard exception message for BDE errors at runtime.

DbErrorIntercept example:

You can call the DbErrorIntercept procedure in the body of your project file (.DPR):

```
begin
   Application.Title := 'My Application';
   DbErrorIntercept;
   Application.CreateForm(TMainForm, MainForm);
   Application.CreateForm(...);
   ...
   Application.Run;
end;
```

UNIT <u>DBExcpt</u> Const <u>DbErrorHelpCtx</u>

Events OnErrorMsg

Routine <u>DbErrorIntercept</u>

Type <u>TDBErrorEvent</u>

Control Name	Unit	Class
RX Picture Clip	<u>PicClip</u>	TPicClip

Description: The TPicClip component provide you the functionality of the PICCLIP.VBX control which ships with Visual Basic Professional. This control allows you to extract portions of a matrix of pictures and assign the "cell picture" to another picture control.

To accomplish this, you might use <u>GraphicCell[Index: Integer]</u> property which extracted a portion of a picture and assigned it to another picture property.

To determine how source picture separate by cells, use Cols and Rows properties. The TPicClip object retrieving "cell picture" via their Index within the range 0 to (Cols * Rows) - 1.

To draw the "cell picture" on a canvas, call the <u>Draw method</u> of a TPicClip object, passing the Canvas as a parameter. The <u>Masked property</u> specifies whether the background of the image obscures objects below the image object when "cell picture" is drawn by the <u>Draw method</u>. <u>MaskColor</u> determines which color of the image is to be transparent when the "cell picture" is drawn.

Property Cols

Declaration: Cols: Integer;

Cols is the number of columns in the matrix of pictures.

Property GraphicCell[Index: Integer]

Declaration: GraphicCell[Index: Integer]: TBitmap;

GraphicCell returns the image ("cell picture") specified by the Index parameter as a TBitmap object. You don't need to free this object later. The TPicClip object retrieving "cell picture" via their Index within the range 0 to (Cols * Rows) - 1, with left-to-right and top-to-bottom direction.

Property MaskColor

Declaration: MaskColor: TColor;

MaskColor determines which color of the "cell picture" is to be transparent when the bitmap is drawn by calling the Draw method and Masked property is True. Use MaskColor to determine how to draw the picture transparently.

Property Masked

Declaration: Masked: Boolean;

Masked specifies whether the background of the "cell picture" obscures objects below the image object when the image is drawn by the Draw method. Default value is True. Set Masked to True to allow objects behind the TPicClip object to show through the background of the bitmap. Set Masked to False to make the background of the "cell picture" opaque.

Property Rows

Declaration: Rows: Integer;

Rows is the number of rows in the matrix of pictures.

Method Draw

Declaration: procedure Draw(Canvas: TCanvas; X, Y, Index: Integer);

Draw renders the "cell picture" specified by the Index parameter on the Canvas at the location given by the coordinates (X, Y). If Masked property is True the MaskColor property determines which color of the "cell picture" is to be transparent when the image is drawn by calling the Draw method.

Method LoadBitmapRes

Declaration: procedure LoadBitmapRes(Instance: THandle; ResID: PChar);

LoadBitmapRes loads the specified bitmap resource along with palette information from a module's executable file or DLL specified by Instance parameter into then TPicClip object. Specify the resource to be loaded as the value of ResID.

Unit <u>PicClip</u> Methods: <u>Draw</u> LoadBitmapRes

Property: <u>Cols</u> <u>GraphicCell[Index: Integer]</u> <u>MaskColor</u> <u>Masked</u> <u>Rows</u>

Control Name	Unit	Class
RX Form Placement	<u>Placemnt</u>	TFormPlacement

Description: The TFormPlacement component provides a quick method for saving size and position of its parent form.

To use the component, specify which properties you want to save in the <u>Options property</u>, and set the <u>IniFileName</u> and <u>IniSections</u> property to the filename of your applications initialization file. In 32-bit version you can set the <u>UseRegistry</u> property to True to use Windows system registry instead of an ini-file.

You can then set property <u>Active</u> to True to allow automatically save and restore specified properties, or call the <u>RestoreFormPlacement</u> and <u>SaveFormPlacement</u> methods to retrieve or save the form. The <u>MinMaxInfo</u> property provides an easy way to limit a form's minimum and maximum tracking size.

Using class **TFormPlacement**

Set <u>Active</u> property to True to enable save and restore parent form's size and position.

When <u>IniFileName</u> is set to the empty string the default name for the INI-file (or Registry Key) will be used.

In 32-bit version, when <u>UseRegistry</u> is True, TFormPlacement uses standard TRegIniFile class. So the IniFileName passed to a TFormPlacement component becomes a subkey under the system registry's root key (HKEY_CURRENT_USER by default).

You can handle the <u>OnSavePlacement</u> ans <u>OnRestorePlacement</u> events to save and restore additional information to the initialization file or Windows Registry.

The TFormPlacement and TFormStorage components provides a quick method for saving size and position of its parent form.

This component also allows control of a parent form with style bsSizeable to have the Minumum and Maximum sizes controlled by the developer.

Property Active Declaration: Active: Boolean;

This property will activate the processing required to save and restore the form's positions.

Property IniFile

Declaration: IniFile: TIniFile;

This property specifies the INI file to use. When this is specified, the form's metric information will be stored inside the INI file, and inside the section.

When <u>IniFileName</u> is set to the empty string the default name for the INI-file (or Registry Key) will be used.

Property IniFileName

Declaration: IniFileName: string;

The IniFileName property is used to tell a saver component the name of the initialization (ini) file it is to write to and read from.

If UseRegistry property is set to True (32-bit version only), then this property specifies Windows system segistry key name.

If you do not specify a value of this property, then default name returned by GetDefaultIniName (or GetDefaultIniRegKey for registry) will be used.

Property IniFileObject

Declaration: IniFileObject: TObject;

This is an IniFileObject. It encapsulates the information so that instead of the information being stored inside the INI file, it is stored inside the registry instead.

(This is used internally)

Property IniSection

Declaration: IniSection: string;

The IniSection is the section of the <u>IniFile</u> which the information will be stored in. If you specify blank, the information about your form's metrics will not be saved.

Property MinMaxInfo

Declaration: MinMaxInfo: TWinMinMaxInfo;

This property allows control of a parent form with style bsSizeable to have the minumum and maximum sizes controlled by the developer.

Property Options

Declaration: Options: TPlacementOptions;

The Options property tells the TFormPlacement component which parts of the parent form it should read and write to the specified IniFileName.

These are the possible values that can be included in the Options set:
- fpState - state of form (normal, minimize, maximize);

- fpPosition - Top, Left, Height and Width of form;

- fpActiveControl - current Active Control on the form (value of the ActiveControl property).

Property PreventResize

Declaration: PreventResize: Boolean;

This property prevents the form from being resized. Setting this to true prevents the form from being resized.

Property RegIniFile

Declaration: RegIniFile: TRegIniFile;

This is used internally. It encapsulates the contents of the Ini file to the registry (something like mapping A to B).

Event OnRestorePlacement

Declaration: OnRestorePlacement: TNotifyEvent;

When the screen is restored, this event is triggered.

A good usage is, "your computer hanged on the previous session. The previous contents of your document has been saved to a temporary file. Would you like to restore the contents of the temporary file?"

Event OnSavePlacement

Declaration: OnSavePlacement: TNotifyEvent;

When the screen is being saved, this event is triggered. A good usage is, "your computer is now auto-saving the contents of your document..."

Method RestoreFormPlacement

Declaration: procedure RestoreFormPlacement; virtual;

The RestoreFormPlacement method loads all the properties specified in the related Options property from the initialization file or registry key specified in IniFileName independently of Active property value.

Method SaveFormPlacement

Declaration: procedure SaveFormPlacement; virtual;

The SaveFormPlacement method saves all the properties specified in the related Options property from the initialization file or registry key specified in IniFileName independently of Active property value.

Property UseRegistry : boolean

This property is enabled under Delphi 2.0 It maps the INI file to the registry instead.

Class TWinMinMaxInfo

This class contains the metrics of the form. It is used by formplacement's <u>options</u> to store information about the form.

It encapsulates: Property MaxPosLeft Property MaxPosTop Property MaxSizeHeight Property MaxSizeWidth Property MaxTrackHeight Property MinMaxInfo Property MinTrackHeight Property MinTrackHeight Property MinTrackHeight

Property MaxPosLeft

Declaration: MaxPosLeft: Integer;

The MinTrackHeight and MinTrackWidth properties specifies the minimum tracking width and the minimum tracking height of the form at run-time.

The MaxTrackHeight and MaxTrackWidth properties specifies the maximum tracking width and the maximum tracking height of the form.

The MaxSizeHeight and MaxSizeWidth properties specifies the maximized width and the maximized height of the form.

The MaxPosLeft è MaxPosTop properties specifies the position of the left side of the maximized window and the position of the top of the maximized window.

Note: Setting any of the above properties to 0 is the same as not constraining by the given axis.

So, setting MinTrackHeight to 0 lets the form be any height (up to whatever MaxSizeHeight is, unless, that to is zero).

Property MaxPosTop

Declaration: MaxPosTop: Integer;

The MinTrackHeight and MinTrackWidth properties specifies the minimum tracking width and the

minimum tracking height of the form at run-time.

The MaxTrackHeight and MaxTrackWidth properties specifies the maximum tracking width and the maximum tracking height of the form.

The MaxSizeHeight and MaxSizeWidth properties specifies the maximized width and the maximized height of the form.

The MaxPosLeft è MaxPosTop properties specifies the position of the left side of the maximized window and the position of the top of the maximized window.

Note: Setting any of the above properties to 0 is the same as not constraining by the given axis.

So, setting MinTrackHeight to 0 lets the form be any height (up to whatever MaxSizeHeight is, unless, that to is zero).

Property MaxSizeHeight

Declaration: MaxSizeHeight: Integer;

The MinTrackHeight and MinTrackWidth properties specifies the minimum tracking width and the minimum tracking height of the form at run-time.

The MaxTrackHeight and MaxTrackWidth properties specifies the maximum tracking width and the maximum tracking height of the form.

The MaxSizeHeight and MaxSizeWidth properties specifies the maximized width and the maximized height of the form.

The MaxPosLeft è MaxPosTop properties specifies the position of the left side of the maximized window and the position of the top of the maximized window.

Note: Setting any of the above properties to 0 is the same as not constraining by the given axis.

So, setting MinTrackHeight to 0 lets the form be any height (up to whatever MaxSizeHeight is, unless, that to is zero).

Property MaxSizeWidth

Declaration: MaxSizeWidth: Integer;

The MinTrackHeight and MinTrackWidth properties specifies the minimum tracking width and the minimum tracking height of the form at run-time.

The MaxTrackHeight and MaxTrackWidth properties specifies the maximum tracking width and the maximum tracking height of the form.

The MaxSizeHeight and MaxSizeWidth properties specifies the maximized width and the maximized height of the form.

The MaxPosLeft è MaxPosTop properties specifies the position of the left side of the maximized window and the position of the top of the maximized window.

Note: Setting any of the above properties to 0 is the same as not constraining by the given axis.

So, setting MinTrackHeight to 0 lets the form be any height (up to whatever MaxSizeHeight is, unless, that to is zero).

Property MaxTrackHeight

Declaration: MaxTrackHeight: Integer;

The MinTrackHeight and MinTrackWidth properties specifies the minimum tracking width and the minimum tracking height of the form at run-time. The MaxTrackHeight and MaxTrackWidth properties specifies the maximum tracking width and the maximum tracking height of the form.

The MaxSizeHeight and MaxSizeWidth properties specifies the maximized width and the maximized height of the form.

The MaxPosLeft è MaxPosTop properties specifies the position of the left side of the maximized window and the position of the top of the maximized window.

Note: Setting any of the above properties to 0 is the same as not constraining by the given axis.

So, setting MinTrackHeight to 0 lets the form be any height (up to whatever MaxSizeHeight is, unless, that to is zero).

Property MaxTrackWidth

Declaration: MaxTrackWidth: Integer;

The MinTrackHeight and MinTrackWidth properties specifies the minimum tracking width and the minimum tracking height of the form at run-time.

The MaxTrackHeight and MaxTrackWidth properties specifies the maximum tracking width and the maximum tracking height of the form.

The MaxSizeHeight and MaxSizeWidth properties specifies the maximized width and the maximized height of the form.

The MaxPosLeft è MaxPosTop properties specifies the position of the left side of the maximized window and the position of the top of the maximized window.

Note: Setting any of the above properties to 0 is the same as not constraining by the given axis.

So, setting MinTrackHeight to 0 lets the form be any height (up to whatever MaxSizeHeight is, unless, that to is zero).

Property MinTrackHeight

Declaration: MinTrackHeight: Integer;

The MinTrackHeight and MinTrackWidth properties specifies the minimum tracking width and the minimum tracking height of the form at run-time.

The MaxTrackHeight and MaxTrackWidth properties specifies the maximum tracking width and the maximum tracking height of the form.

The MaxSizeHeight and MaxSizeWidth properties specifies the maximized width and the maximized height of the form.

The MaxPosLeft è MaxPosTop properties specifies the position of the left side of the maximized window and the position of the top of the maximized window.

Note: Setting any of the above properties to 0 is the same as not constraining by the given axis.

So, setting MinTrackHeight to 0 lets the form be any height (up to whatever MaxSizeHeight is, unless, that to is zero).

Property MinTrackWidth

Declaration: MinTrackWidth: Integer;

The MinTrackHeight and MinTrackWidth properties specifies the minimum tracking width and the minimum tracking height of the form at run-time.

The MaxTrackHeight and MaxTrackWidth properties specifies the maximum tracking width and the maximum tracking height of the form.

The MaxSizeHeight and MaxSizeWidth properties specifies the maximized width and the maximized height of the form.

The MaxPosLeft è MaxPosTop properties specifies the position of the left side of the maximized window and the position of the top of the maximized window.

Note: Setting any of the above properties to 0 is the same as not constraining by the given axis.

So, setting MinTrackHeight to 0 lets the form be any height (up to whatever MaxSizeHeight is, unless, that to is zero).

Method DefaultMinMaxInfo

Declaration: function DefaultMinMaxInfo: Boolean;

Returns True when all key properties of TWinMinMaxInfo object are set to 0. For internal using only.

Type TPlacementOption

Declaration: TPlacementOption = (fpState, fpPosition, fpActiveControl);

The TPlacementOption type defines the possible values for the Options property of the TFormPlacement or TFormStorage components.

Type TPlacementOptions

-

Declaration: TPlacementOptions = set of TPlacementOption;

The TPlacementOption type defines the possible values for the Options property of the TFormPlacement or TFormStorage components.

UNIT **Placement**

Class <u>Using class TFormStorage</u>

Property Active Options StoredProps

Method <u>RestoreProperties</u> <u>SaveProperties</u>

<u>Class</u> <u>Using class TFormPlacement</u>

Event OnRestorePlacement OnSavePlacement

Method DefaultMinMaxInfo RestoreFormPlacement SaveFormPlacement

Property Active IniFile IniFileName IniFileNameObject IniSection Options PreventResize RegIniFile UseRegistry : boolean

Type <u>TPlacementOption</u>

Class TWinMinMaxInfo

Property <u>MaxPosLeft</u> <u>MaxPosTop</u> MaxSizeHeight MaxSizeWidth MaxTrackHeight MaxTrackWidth MinMaxInfo MinTrackHeight MinTrackHeight MinTrackWidth

Control Name	Unit	Class
RX Form Storage	<u>Placement</u>	TFormStorage

Description:

The TFormStorage component is a direct descendent of TFormPlacement. In addition to the

TFormPlacement's properties, methods, and events, this component also allows a component to save its any published property values off to an INI file (or 32-bit registry) and restore the values later.

In 32-bit version, when UseRegistry is True, TFormStorage uses standard TRegIniFile class. So the IniFileName passed to a TFormStorage component becomes a subkey under the system registry's root key (HKEY_CURRENT_USER by default).

TFormStorage will work with any component or control you might have. Even 3rd party controls or controls you have designed yourself.

Using class TFormStorage

- At design-time use component editor to add published properties of any other components to list of stored properties of TFormStorage component.
- To save and restore the position of TRxSplitter component add to stored properties the Width or Height (according to Align property) of one of splitted controls (ControlFirst or ControlSecond).
- To save and restore active page of tabbed controls (TNotebook, TTabbedNotebook or TPageControl) store ActivePage or PageIndex property.
- You can save and restore text of edit controls by adding to stored properties the Text property for TEdit or Lines property for TMemo.
- Use TFormStorage to save and restore state of windowed controls such as TCheckBox, TRadioGroup etc.
- Link the TFormStorage (or TFormPlacement) component to IniStorage property of some another components to save and load information specific for these components (TRxCheckListBox, TRxDrawGrid, TSpeedbar, TMRUManager, TRxDBGrid) with no additional code.

TFormStorage component allows you to read and write virtually any component published property to an INI file or the system Registry with virtually no code.

Property Active

Declaration: Active: Boolean;

Property active defines whether FormStorage service is active or not.

Property Options

Declaration: Options: TPlacementOptions;

The Options property tells the TFormPlacement component which parts of the parent form it should read and write to the specified IniFileName.

These are the possible values that can be included in the Options set:

- fpState state of form (normal, minimize, maximize);
- fpPosition Top, Left, Height and Width of form;
- fpActiveControl current Active Control on the form (value of the ActiveControl property).

Property StoredProps

Declaration: StoredProps: TStrings;

StoredProps contains the forms controls you have selected or chosen to have their properties stored into a file.

Method RestoreProperties

Declaration: procedure RestoreProperties;

procedure RestoreProperties restores the properties of the selected controls and form.

Method SaveProperties

Declaration: procedure SaveProperties;

procedure SaveProperties save the properties of the selected controls and form.

Control Name	Unit	Class
TRxIniFile	<u>RxIni</u>	TRxIniFile

Description:

RxIniFile is an enhanced version of the IniFile specially designed to store <u>FormStorage</u> and <u>Rx Form</u> <u>Placement</u>.

Property ListItemName

Declaration: ListItemName: string;

ListItemName is the item's list name.

Event OnReadObject

Declaration: OnReadObject: TReadObjectEvent;

OnReadObject event is triggered when the data is read.

Event OnWriteObject

Declaration: OnWriteObject: TWriteObjectEvent;

OnWriteObject event is triggered when the data is written into the INI file.

Method ReadClearList

Declaration: function ReadClearList(const Section: string; List: TStrings): TStrings;

Function ReadClearList reads a string list from a section.

Method ReadColor

Declaration: function ReadColor(const Section, Ident: string; Default: TColor): TColor;

Method ReadColour allows a colour to be read from the particular section and identification part of an INI file.

Method ReadFont

Declaration: function ReadFont(const Section, Ident: string; Font: TFont): TFont;

Method ReadFont reads font details from the Section and Ident of an INI file.

Method ReadList

Declaration: function ReadList(const Section: string; List: TStrings): TStrings;

Method ReadList reads a list from a section of the INI file.

Method ReadPoint

Declaration: function ReadPoint(const Section, Ident: string; const Default: TPoint): TPoint;

Method ReadPoint reads a point from the Inifile, from Section and Ident.

Method ReadRect

Declaration: function ReadRect(const Section, Ident: string; const Default: TRect): TRect;

Method ReadRect reads the *value* of TRect to a section and Ident of the specified IniFile.

Method WriteColor

Declaration: procedure WriteColor(const Section, Ident: string; Value: TColor);

Method WriteColor writes the *value* of the colour to a *section* and *ident* of the specified IniFile.

Method WriteFont

Declaration: procedure WriteFont(const Section, Ident: string; Font: TFont);

Method WriteFont writes the *value* of the font to a section and identifier of the specified IniFile.

Method WriteList

Declaration: procedure WriteList(const Section: string; List: TStrings);

Method WriteList writes the *value* of the string list to a section of the specified IniFile.

Method WritePoint

Declaration: procedure WritePoint(const Section, Ident: string; const Value: TPoint);

Method WritePoint writes the *value* of TPoint to a section and Ident of the specified IniFile.

Method WriteRect

Declaration: procedure WriteRect(const Section, Ident: string; const Value: TRect);

Method WriteRect writes the *value* of TRect to a section and Ident of the specified IniFile.

Type TReadObjectEvent

Declaration: TReadObjectEvent = function(Sender: TObject; const Section, Item, Value: string): TObject of object;

TReadObjectEvent is the type of the OnReadObject event of the TRxIniFile class.

Type TWriteObjectEvent

Declaration: TWriteObjectEvent = procedure(Sender: TObject; const Section, Item: string; Obj: TObject) of object;

TWriteObjectEvent is the type of the OnWriteObject event of the TRxIniFile class.

UNIT <u>RxIni</u>

Property ListItemName

Event OnReadObject OnWriteObject

Method ReadClearList ReadColor ReadFont ReadList ReadPoint ReadRect WriteColor WriteFont WriteList WritePoint WriteRect

Type <u>TReadObjectEvent</u> <u>TWriteObjectEvent</u>

Control Name	Unit	Class
RX Application Events	<u>RxHook</u>	TRxWindowHook

Description:

This class allows the creation of a "windows system hook". In other words, register an event with the windows event system so that it can be processed by your application instead. As it's name implies, it is not easy to understand what it does.

In other words, if you do not have the source code Make an extension to a component that does not have the source code, or is difficult to modify.

Property WinControl

Declaration: WinControl: TWinControl;

This is the component to "hook" to.

Event AfterMessage

Declaration: AfterMessage: THookMessageEvent;

AfterMessage event is triggered after a message generated from that particular control is passed.

Event BeforeMessage

Declaration: BeforeMessage: THookMessageEvent;

BeforeMessage event is triggered before a message is generated from that particular control being monitored.

Type THookMessageEvent

Declaration: THookMessageEvent = procedure (Sender: TObject; var Msg: TMessage; var Handled: Boolean) of object;

THookMessageEvent is the type of the BeforeMessage and AfterMessage events of the TRxWindowHook component. The Handled parameter have a sence only in the BeforeMessage event.

Routine FindVirtualMethodIndex

Declaration: function FindVirtualMethodIndex(AClass: TClass; MethodAddr: Pointer): Integer;

FindVirtualMethodIndex example:

type THookForm = class(TForm)
protected
procedure WndProc(var Message: TMessage); override;
end;

$\{\ldots\}$

WndProcSeq := FindVirtualMethodIndex(THookForm, @THookForm.WndProc);
{...}

Routine GetVirtualMethodAddress

Declaration: function GetVirtualMethodAddress(AClass: TClass; AIndex: Integer): Pointer;

GetVirtualMethodAddress example:

Method := GetVirtualMethodAddress(TForm1, 1);

Routine SetVirtualMethodAddress

Declaration: function SetVirtualMethodAddress(AClass: TClass; AIndex: Integer; NewAddress: Pointer): Pointer;

SetVirtualMethodAddress hooks the custom control *AClass* and specific type *AIndex* and linking it to *new address*.

SetVirtualMethodAddress example: Next example shows how you can replace address of virtual method WndProc for descendant of TForm:

{ THookForm }

```
type
THookForm = class(TForm)
protected
procedure WndProc(var Message: TMessage); override;
end;
```

```
procedure THookForm.WndProc(var Message: TMessage);
var I: Integer;
begin
    inherited WndProc(Message);
    if (Message.Msg = WM_NCHITTEST) then
    try
        { your specific event handler }
    except
        Application.HandleException(Self);
    end;
end;
```

Replace and restore method address:

type PClass = ^TClass; var SaveMethod: TWndMethod;

procedure SetHook(AForm: TForm); var MethodIndex: Integer;

```
begin
    MethodIndex := FindVirtualMethodIndex(THookForm, @THookForm.WndProc);
    @SaveMethod := SetVirtualMethodAddress(PClass(Form)^,
    MethodIndex, @THookForm.WndProc);
end;
procedure ReleaseHook(AForm: TForm);
var MethodIndex: Integer;
begin
    if Assigned(SaveMethod) then
    begin
     MethodIndex := FindVirtualMethodIndex(THookForm, @THookForm.WndProc);
     SetVirtualMethodAddress(PClass(Form)^, MethodIndex, @SaveMethod);
     SaveMethod := nil;
    end;
end;
```

```
Index Page | About | Download
Creation Date: 4 Feb 1998 | Last Update: 16 Mar 2000
```

UNIT <u>RxHook</u> Property <u>WinControl</u>

Event <u>AfterMessage</u> <u>BeforeMessage</u>

Type <u>THookMessageEvent</u>

Routine FindVirtualMethodIndex GetVirtualMethodAddress SetVirtualMethodAddress

Control Name	Unit	Class
RX Dual List Dialogue	<u>DualList</u>	TDualListDialog

Description:

DualListDialogue is a window with two listboxes, one is the source, the other is the destination. If you want to change the captions of the Help, Cancel and OK, use the CancelBtnCaption properties.

Property CancelBtnCaption

Declaration: CancelBtnCaption: TCaption;

Property CancelBtnCaption sets the caption of the Cancel Button.

Property HelpBtnCaption

Declaration: HelpBtnCaption: TCaption;

Property HelpBtnCaption sets the caption of the Help Button.

Property Label1Caption

Declaration: Label1Caption: TCaption;

Property Label1Caption ("Source") sets the caption of the Help Button.

Property Label2Caption

Declaration: Label2Caption: TCaption;

Property Label2Caption ("Destination") sets the caption of the Help Button.

Property List1

Declaration: List1: TStrings;

Property List1 is the source container. List1 contain the strings which would be transferred to List2.

Property List2

Declaration: List2: TStrings;

Property List2 is the target container. The items in List1 are transferred to List2.

Property OkBtnCaption

Declaration: OkBtnCaption: TCaption;

Property OkBtnCaption ("Ok") sets the caption of the Ok Button.

Property ShowHelp

Declaration: ShowHelp: Boolean;

ShowHelp determines whether the "Help" button is shown or not.

Property Sorted Declaration: Sorted: Boolean;

Property Sorted sorts the list of the dual lists.

Property Title Declaration: Title: string;

The Title property determines the text that appears in the dialog box's title bar.

Index Page | About | Download Creation Date: 4 Feb 1998 | Last Update: 16 Mar 2000 - 1

UNIT DualList

Property CancelBtnCaption HelpBtnCaption Label1Caption List1 List2 OkBtnCaption ShowHelp Sorted Title

RX Converter

UNIT DataConv

Class TConverter

This component allows you to access data of any types and convert data between types. TConverter class has properties AsBoolean, AsDate, AsDateTime, AsFloat, AsInteger, AsString and AsTime. Use these properties as appropriate to access or modify the current value stored in TConverter object.

Property AsBoolean

Declaration: AsBoolean: Boolean;

Run-time only. This is a conversion property. AsBoolean can be used to read or set the value as a boolean value (True or False). The DisplayFalse and DisplayTrue properties controls the manner in which boolean values are translated to and from string format by the TConverter component.

Property AsDate

Declaration: AsDate: TDateTime;

Run-time only. This is a conversion property. AsDate can be used to read or set the value stored in the TConverter component as a date value.

Property AsDateTime

Declaration: AsDateTime: TDateTime;

Run-time only. This is a conversion property. AsDateTime can be used to read or set the value stored in the TConverter component as a date-and-time value.

Property AsFloat

Declaration: AsFloat: Double;

Run-time only. This is a conversion property. AsFloat can be used to read or set the value stored in the TConverter component as a Double.

Property AsInteger

Declaration: AsInteger: Longint;

Run-time only. This is a conversion property. AsInteger can be used to read or set the value stored in the TConverter component as an intheger value.

Property AsString

Declaration: AsString: string;

Run-time only. This is a conversion property. AsString can be used to read or set the value stored in the TConverter component as a string value.

When then DataType property is not equal to dtString value and RaiseOnError is True, an exception is raised if TConverter component can't convert a string to the appropriate type.

Property AsTime

Declaration: AsTime: TDateTime;

Run-time only. This is a conversion property. AsTime can be used to read or set the value stored in the TConverter component as a time value.

Property DataType

Declaration: DataType: TDataType;

The DataType identifies the data type of the value stored in the TConverter component.

Property DateTimeFormat

Declaration: DateTimeFormat: TDateTimeFormat;

The value of this property controls the format of date-and-time values when TConverter component converts date-time data to string and backward.

Property Digits

Declaration: Digits: Integer;

Digits is the number of digits that are displayed to the right of the decimal point.

Property DisplayFalse

Declaration: DisplayFalse: string;

The DisplayFalse and DisplayTrue properties controls the manner in which boolean values are translated to and from string format by the TConverter component.

Property DisplayTrue

Declaration: DisplayTrue: string;

The DisplayFalse and DisplayTrue properties controls the manner in which boolean values are translated to

and from string format by the TConverter component.

Property FloatFormat

Declaration: FloatFormat: TFloatFormat;

The value of this property controls the format of float values when TConverter component converts float data to string and backward.

Property Precision

Declaration: Precision: Integer;

The Precision property is used in formatting float fields. The value of Precision is the number of decimal places to the right of the decimal point the numeric value should be formatted to before rounding begins. The default value is 15 decimal places.

Property RaiseOnError

Declaration: RaiseOnError: Boolean;

This property determines whether or not an exception is raised when error of type conversion is occured. Default value is True.

Property Text Declaration: Text: string;

This property can be used to set value stored in the TConverter component as a string value. Available at design-time and at run-time.

Method Clear Declaration:

procedure Clear;

This method clears the value stored in TConverter component.

Method IsValidChar

Declaration:

function IsValidChar(Ch: Char): Boolean; virtual;

IsValidChar is used by TConverter component to determine if a particular character is valid for the type specified by DataType property.

Class TDateTimeFormat

TDateTimeFormat class used by TConverter component for formatting the date-and-time values.

Property AMString

Declaration: AMString: string;

Character string used to indicate morning (before noon and after midnight) times, when using a twelve-hour clock. Default value is standard TimeAMString constant from SYSUTILS.PAS unit.

Property DateMask Declaration: DateMask: string;

The DateMask (read-only) property specifies the date output format. The value of this property corresponds to the values of other properties of TDateTimeFormat object.

Property DateOrder

Declaration: DateOrder: TDateOrder;

The DateOrder property controls the order of the month, day, and year components of date. The default is the order normally used in the country selected in the Windows Control Panel.

Property DateSeparator

Declaration: DateSeparator: Char;

The character used to separate the year, month, and day parts of a date value. The initial value is correspond with DateSeparator constant from standard SYSUTILS.PAS module.

Property FourDigitYear

Declaration: FourDigitYear: Boolean;

Specifies the number of digits for the year value (four or two). If True, years are expressed in four digits (i.e., 1997). If False, years have two digits (97).

Property LeadingZero

Declaration: LeadingZero: Boolean;

Specifies whether or not single digit month or day values have a leading zero. For example, if you enter "1/1/80" and this is set to TRUE, TDateTimeFormat interprets the date as "01/1/80." If FALSE, the value is "1/1/80." Default: TRUE.

Property LongDate

Declaration: LongDate: Boolean;

When this property is True, then the full month's names (January-December, using the strings given by the LongMonthNames global variable) are using for output. When False, displays the month as a number (1-12).

Property Mask

Declaration: Mask: string;

The Mask (read-only) property specifies the date-and-time output format. The value of this property corresponds to the values of other properties of TDateTimeFormat object.

Property PMString

Declaration: PMString: string;

Character string used to indicate evening (after noon and before midnight) times, when using a twelve-hour clock. Default value is standard TimePMString constant from SYSUTILS.PAS unit.

Property TimeFormat

Declaration: TimeFormat: TTimeFormat;

The TimeFormat property controls the order and number of digits in the hours, minutes, and seconds components of time.

Property TimeMask

Declaration: TimeMask: string;

The TimeMask (read-only) property specifies the time output format. The value of this property corresponds to the values of other properties of TDateTimeFormat object.

Property TimeSeparator

Declaration: TimeSeparator: Char;

The character used to separate the hour, minute, and second parts of a time value. The initial value is fetched from the TimeSeparator variable in the SYSUTILS.PAS standard unit.

Method ResetDefault

Declaration: procedure ResetDefault; virtual;

ResetDefault clears the values of all properties of TDateTimeFormat object and sets them to its default values. Default values for these properties are depends on Windows settings.

Type TDataType

Declaration: TDataType = (dtString, dtInteger, dtFloat, dtDateTime, dtDate, dtTime, dtBoolean);

TDataType is a set of values for the DataType property of the TConverter component.

Type TTimeFormat

Declaration: TTimeFormat = (tfHHMMSS, tfHMMSS, tfHHMM, tfHMM);

TTimeFormat is a set of values for the TimeFormat property of the TDateTimeFormat class.

UNIT DataConv

Property AsBoolean <u>AsDate</u> **AsDateTime** <u>AsFloat</u> <u>AsInteger</u> AsString AsTime **DataType** DateTimeFormat **Digits DisplayFalse DisplayTrue FloatFormat** Precision **RaiseOnError** Text

Method <u>Clear</u> <u>IsValidChar</u>

Class TDateTimeFormat AMString DateMask DateOrder DateSeparator FourDigitYear LeadingZero LongDate Mask PMString TimeFormat TimeMask TimeSeparator

Method <u>ResetDefault</u>

Type <u>TDataType</u> <u>TTimeFormat</u>

Control Name	Unit	Class
RX Secret Panel	<u>RXCtrls</u>	TSecretPanel

Description:

TSecretPanel implements the generic behavior introduced in TPanel and publishes all properties inherited from TPanel. TSecretPanel also allows lines of text (Lines property) to be scrolled vertically from bottom to top when the <u>Active property</u> is True. It facilitates, amoung other things, the text display features to be changed, such as alignment, font and style (lowered, raised, or normal). You can also display graphic specified by <u>Glyph property</u> during text scrolling. When <u>Active</u> is False the lines of text and image Glyph are invisible.

In other words, this panel acts as a credits panel, or a scrolling text panel, where the text is scrolled similar to movie credits.

A good usage of this panel is an enhanced version of the about box. Use the <u>Glyph property</u> to set the bitmap and the <u>GlyphLayout</u> to set the left part of the panel. Set the <u>Active</u> property and <u>Cycled</u> property to true, and add the text of the property <u>lines</u>.

Property Active

Declaration: Active: Boolean;

Active property defines whether the scrolling is active or not. You must have something defined in the property <u>lines</u>.

Property Cycled

Declaration: Cycled: Boolean;

Cycles property defines whether Active is property should be active after the text scrolled out of the SecretPanel.

In other words, once the text in the <u>lines property</u> are scrolled out of the SecretPanel, <u>Active</u> becomes false. To set <u>Active</u> to permanently, set the <u>Cycled property</u> to true.

Property GlyphLayout

Declaration: GlyphLayout: TGlyphLayout;

GlyphLayout property sets the position of the Glyph.

Property Glyph

Declaration: Glyph: TBitmap;

Glyph property sets the Glyph's bitmap.

Property Lines

Declaration: Lines: TStrings;

The lines property contains the lines that will be scrolled

Property TextStyle

Declaration: TextStyle: TPanelBevel;

TextStyle sets the style of the text -

- bvLowered sub-raised text.
- bvNone none
- bvRaised raised text.

Event OnStartPlay

Declaration: OnStartPlay: TNotifyEvent;

OnStartPlay event is triggered when the <u>Active</u> property is set to true.

Event OnStopPlay

Declaration: OnStopPlay: TNotifyEvent;

OnStartPlay event is triggered when the <u>Active</u> property is set to false.

Control Name	Unit	Class
RX Speed Bar	<u>Speedbar</u>	TSpeedbar

Description:

TSpeedBar manages speed buttons, arranging them in rows and automatically adjusting their sizes and positions.

TSpeedBar is a container for speed buttons (TSpeedItem). It provides an easy way to arrange and manage TSpeedButton controls.

• All speed buttons on a speed bar maintain a uniform width, height and style.

• The Options property allows the background to show through the speed buttons and gives pop-up borders to the speed buttons.

• Spaces and dividers (which are in fact specially configured speed buttons) can group buttons on the TSpeedBar both visually and functionally.

Typically, the speed buttons correspond to items in an application's menu and give the user more direct access to the application's commands.

Speedbar design-time component editor provides a method to create and edit your speedbar and buttons. TSpeedbar has a methods to bring up Customize Dialog at run-time and user can customize the speed bar in run-time using drag-and-drop operation, similar to Delphi speedbar.

TSpeedbar has also methods to store and load layout using ini-file or Win'95 system registry.

Property BoundLines

Declaration: BoundLines: TBoundLines;

Use this property to specify which sides of the Speedbar component will be framed with a recessed (lowered) line.

Property BtnHeight

Declaration: BtnHeight: Integer;

BtnHeight represents the height, in pixels, of the buttons in the TSpeedBar. During upgrading your application, don't remember to increase value of Version property if the BtnHeight or BtnWidth property has been changed.

Property BtnOffsetHorz

Declaration: BtnOffsetHorz: Integer;

Determines the minimum available horizontal offset (left coordinate) for the left-most buttons in Speedbar.

Property BtnOffsetVert

Declaration: BtnOffsetVert: Integer;

Determines the minimum available vertical offset (top coordinate) for the top-most buttons in Speedbar.

Property BtnWidth

Declaration: BtnWidth: Integer;

BtnWidth represents the width, in pixels, of the buttons in the TSpeedBar. During upgrading your application, don't remember to increase value of Version property if the BtnHeight or BtnWidth property has been changed.

Property IniStorage

Declaration: IniStorage: TFormPlacement;

IniStorage property is the IniStorage component that this component saves and restores all the component state information from.

Property Options

Declaration: Options: TSpeedbarOptions;

TSpeedbarOption = (sbAllowDrag, sbAllowResize, sbFlatBtns, sbGrayedBtns, sbTransparentBtns, sbStretchBitmap);

- sbAllowDrag allow the speed bar to be dragged to a new position.
- sbAllowResize allows the speed bar to be resized.
- sbFlatBtns subclasses the <u>RxSpeedButtons</u> so that they appear to be flat.
- sbGrayedBtns subclasses the <u>RxSpeedButtons</u> so that they appear grayed.
- sbTransparentBtns subclasses the <u>RxSpeedButtons</u> so that they appear transparent.
- sbStretchBitmap stretches the wallpaper if necessary.

Property Position

Declaration: Position: TBarPosition;

TBarPosition = (bpAuto, bpCustom);

Positition subclasses the position so that it appears to be either *automatically located* (bpAuto) or customised to some certain configuration (bpCustom).

Property Sections Declaration: Sections: TList; Sections are sections of speedbar so Speedbuttons can be added in a systematic manner.

Property Version Declaration: Version: Integer;

Apparently does nothing.

Property Wallpaper

Declaration: Wallpaper: TPicture;

WallPaper property contains the picture that is used to "wallpaper" or cover the speedbar.

Event OnApplyAlign

Declaration: OnApplyAlign: TApplyAlignEvent;

OnApplyAlign sets the alignment to a particular part of the form [top, bottom, left and right]. OnApplyAlign is useful for setting some settings that are form specific, e.g., the position of the speedbuttons (from Left-right to Top bottom).

Event OnApplyAlign example

procedure TMainForm.SpeedBarApplyAlign(Sender: TObject; Align: TAlign; var Apply: Boolean); begin Apply := Align in [alTop, alBottom];

Apply := Align in [allop, alBottom]; end;

Event OnCustomize

Declaration: OnCustomize: TNotifyEvent;

OnCustomize occurs when the speedbar's Customize Dialog closes. Write an OnCustomize event handler to perform special processing when the customize dialog closes.

Event OnPosChanged

Declaration: OnPosChanged: TNotifyEvent;

OnPosChange event is triggered when the position of the speedbar has changed.

Event OnVisibleChanged

Declaration: OnVisibleChanged: TNotifyEvent;

OnVisibleChanged occurs when the control is shown or hide (that is, when the control's Visible property changes).

Method AddItem

Declaration: procedure AddItem(Section: Integer; Item: TSpeedItem);

AddItem procedure adds an *item* speedbar item to a particular section.

Method AddSection

Declaration: function AddSection(const ACaption: string): Integer;

AddSection adds a particular section to the toolbar.

Method Customize

Declaration: procedure Customize(HelpCtx: Longint);

Customize sets the helpc context number.

Method FindItem

Declaration: function FindItem(Item: TSpeedItem; var Section, Index: Integer): Boolean;

FindItem tries to find *item*. If the function returns true, the *Section* and *Index* would be filled with the appropriate IDs where exactly the item is.

Method Items

Declaration: function Items(Section, Index: Integer): TSpeedItem;

Items function returns the *speeditem* based on the *section* id and the index of the speeditem.

Method ItemsCount

Declaration: function ItemsCount(Section: Integer): Integer;

ItemsCount function counts the number of items on a speedbar's section.

Method RemoveItem

Declaration: procedure RemoveItem(Item: TSpeedItem);

RemoveItem procedure removes the *item* from the speedbar.

Method RemoveSection

Declaration: procedure RemoveSection(Section: Integer);

RemoveSection procedure removes the section based on the section number.

Method SearchSection

Declaration: function SearchSection(const ACaption: string): Integer;

SearchSection function searches for a particular *caption* on the speedbar.

Method SetFontDefault

Declaration: procedure SetFontDefault;

SetFontDefault sets the font to the default font.

Class TSpeedItem

TSpeedItem is a wrapper component for the button control (TRxSpeedButton) in a TSpeedBar object. To place speed buttons on a speed bar at design time, select the speed bar, double-click, and choose Add. All buttons in the speed bar can be grouped by the sections. You can rearange the buttons in a speed bar at design time and at runtime by using drag-and-drop operations while the Speedbar Designer is shown.

Property BtnCaption

Declaration: BtnCaption: TCaption;

BtnCaption specifies a text string that will be shown as a caption of button control (TRxSpeedButton) in the speed bar.

Property Button

Declaration: Button: TRxSpeedButton;

Run-time and read-only property to direct access to the button in a speed-item component as to the TRxSpeedButton control.

Property Stored

Declaration: Stored: Boolean;

This property determines whether or not button's position and visibility will be stored in an INI-file (or in a system registry) when application calls TSpeedbar.SaveLayout (or SaveLayourReg) method.

Method ButtonClick Declaration: procedure ButtonClick;

ButtonClick simulates a mouse click, as if the user had clicked the button of TSpeedItem.

Type TApplyAlignEvent

Declaration: TApplyAlignEvent = procedure (Sender: TObject; Align: TAlign; var Apply: Boolean) of object;

TApplyAlignEvent is the type of the OnApplyAlign event of the TSpeedBar component.

Type TBarOrientation

Declaration: TBarOrientation = (boHorizontal, boVertical);

TBarOrientation is a set of values for the Orientation property of the TSpeedbar component.

Type TBarPosition

Declaration: TBarPosition = (bpAuto, bpCustom);

TBarPosition is a set of values for the Position property of the TSpeedbar component.

Type TBoundLine

Declaration: TBoundLine = (blTop, blBottom, blLeft, blRight);

TBoundLines is a set of values for the BoundLines property of the TSpeedbar component.

Type TBoundLines

Declaration: TBoundLines = set of TBoundLine;

TBoundLines is a set of values for the BoundLines property of the TSpeedbar component.

Type TSpeedbarOption

Declaration: TSpeedbarOption = (sbAllowDrag, sbAllowResize, sbFlatBtns, sbGrayedBtns, sbTransparentBtns, sbStretchBitmap);

TSpeedbarOptions is a set of values for the Options property of the TSpeedbar component.

Type TSpeedbarOptions

Declaration: TSpeedbarOptions = set of TSpeedbarOption;

TSpeedbarOptions is a set of values for the Options property of the TSpeedbar component.
Routine FindSpeedBar

Declaration: function FindSpeedBar(const Pos: TPoint): TSpeedBar;

FindSpeedBar attempts to find if the speedbar is at position X and Y.

```
FindSpeedBar example:
  Bar := FindSpeedBar(Point(X, Y));
  if Bar <> nil then
  begin
    {...}
  end;
```

UNIT SpeedBar

Property BoundLines BtnHeight BtnOffsetHorz BtnOffsetVert BtnWidth IniStorage Options Orientation Position Sections Version Wallpaper

Event

OnApplyAlign OnCustomize OnPosChanged OnVisibleChanged

Method

AddItem AddSection Customize FindItem Items ItemsCount RemoveItem RemoveSection SearchSection SetFontDefault

Class <u>TSpeedItem</u>

Property BtnCaption Button Stored

Method ButtonClick

Type TApplyAlignEvent TBarOrientation TBarPosition TBoundLine <u>TBoundLines</u> <u>TSpeedbarOption</u> <u>TSpeedbarOptions</u>

Routine <u>FindSpeedBar</u>

Control Name	Unit	Class
RX Clip Board Viewer	<u>ClipView</u>	TClipboardViewer

Description: TClipBoardViewer is a special panel that can display the contents of the Windows Clipboard on a form. This is useful for examining the contents of the Windows clipboard without cutting & pasting.

As the clipboard stores information in multiple formats, you can use this to feature to transfer information between programs that use different formats.

The <u>ViewFormat</u> property allows you to change current display format of Clipboard contents. The ClipboardFormatNames indexed property (run-time and read-only) is a list of the formats available for the information currently on the Clipboard.

TClipboardViewer component automatically changes <u>ViewFormat</u> property when Clipboard contents is changed. You can set format you needed in your handler of <u>OnChange</u> event.

Use TClipboard class to determine formats currently available on the Clipboard.

Property ClipboardFormatNames[Index: Integer]

Declaration: ClipboardFormatNames[Index: Integer]: string;

The ClipboardFormatNames indexed property (run-time and read-only) is a list of the formats available for the information currently on the Clipboard. Use FormatCount property of TClipboard class to determine maximum value of index for ClipboardFormatNames.

You can use values of this property, fo example, to display list of currently available clipboard formats or to create menu items with clipboard format names.

Property ViewFormat

Declaration: ViewFormat: TClipboardViewFormat;

The Clipboard stores information in multiple formats so that you can transfer information between programs that use different formats. The ViewFormat property allows you to change current display format of Clipboard contents. TClipboardViewer component automatically changes ViewFormat property when Clipboard contents is changed. You can set format you needed in your handler of OnChange event.

To determine whether or not TClipboardViewer can display specified data format use CanDrawFormat method.

Event OnChange

Declaration: OnChange: TNotifyEvent;

OnChange occurs whenever the contents of the clipboard may have changed. Write an OnChange event handler to take specific action whenever the contents of the clipboard may have changed. For example, you can specify appropriate display format of TClipboardViewer component (ViewFormat property) in this event handler.

Method CanDrawFormat

Declaration: function CanDrawFormat(ClipboardFormat: Word): Boolean;

This class method returns True if data format specified by ClipboardFormat parameter is supported by TClipboardViewer class and can be displayed. Otherwise CanDrawFormat returns False.

Type TClipboardViewFormat

Declaration: TClipboardViewFormat = (cvDefault, cvEmpty, cvUnknown, cvText, cvBitmap, cvMetafile, cvPalette, cvOemText, cvPicture, cvComponent, cvIcon);

The TClipboardViewFormat type defines the possible values for the ViewFormat property of the TClipboardViewer component.

UNIT <u>ClipView</u> Event <u>OnChange</u>

Method <u>CanDrawFormat</u>

Property <u>ClipboardFormatNames[Index: Integer]</u> <u>ViewFormat</u>

Type <u>TClipboardViewFormat</u>

Control Name	Unit	Class
RX Page Manger	PageMngr	TPageManager

Note:

Please see the demo. It is really difficult to explain what the PageManager does in English as most of the operations are taken care of by Borland's tNoteBook.

In other words, PageManger acts as a middle-man between buttons assigned to OkButton, CancelButton, HelpButton and tNoteBook.

Description:

TPageManager gives you the possibility of "one-click" creating "step-by-step" dialogs (known as "experts" or "wizards"). The database form Wizard dialog is a typical example of that. This component has no pages, it uses another page owner defined by PageOwner property. Currently only TNotebook is supported as page owner.TPageManager has OnGetNextPage and OnGetPriorPage events to handle sequence of pages. Also TPageManager has PageProxies property which contains list of page proxies («steps»), each of them has OnEnter, OnShow, OnHide è OnLeave events. These events occur when the user clicks the Back and Next buttons specified by NextBtn and PriorBtn properties. You can also change current page programatically by calling NextPage and PriorPage methods.

Property NextBtn

Declaration: NextBtn: TControl;

NextBtn defines which control to the NextBtn on the form which should be used to control the pages

Property NextEnabled

Declaration: NextEnabled: Boolean;

The variable contained enables or disable the "Next" button defined on the form.

Property PageCount

Declaration: PageCount: Integer;

PageCount is the total number of pages that the PageManager contains.

Property PageIndex

Declaration: PageIndex: Integer;

PageIndex is the total number of pages that the PageManager contains.

Property PageNames[Index: Integer]

Declaration: PageNames[Index: Integer]: string;

PageNames is the current name of the selected page..

Property PageOwner

Declaration: PageOwner: TPageOwner;

This is the notebook component that the MRU manager hooks upon.

Property PageProxies

Declaration: PageProxies: TList;

PageProxies is the list of cached pages that is contained in the notebook.

Property PriorBtn

Declaration: PriorBtn: TControl;

PriorBtn defines which control to the PriorBtn on the form which should be used to control the pages.

Property PriorEnabled

Declaration: PriorEnabled: Boolean;

The variable contained enables or disable the "Prior" button defined on the form.

Property SetStartPage

Declaration: SetStartPage: Boolean;

SetStartPage property sets the start page.

In other words, SetStartPage sets the particular page to start.

Event OnCheckButtons

Declaration: OnCheckButtons: TNotifyEvent;

OnCheckButtons

Event OnGetNextPage

Declaration: OnGetNextPage: TPageRequestEvent;

OnGetNextPage is a tPageRequestEvent

Event OnGetPriorPage

Declaration: OnGetPriorPage: TPageRequestEvent;

OnGetNextPage is a tPageRequestEvent

GetPriorpage

Method GetNextPageIndex Declaration: function GetNextPageIndex(Page: Integer): Integer;

THIS TOPIC IS NOT COMPLETE!

Method GetPriorPageIndex Declaration: function GetPriorPageIndex(Page: Integer): Integer;

THIS TOPIC IS NOT COMPLETE!

Method GetUniqueName

Declaration: function GetUniqueName(Component: TComponent): string;

THIS TOPIC IS NOT COMPLETE!

Method NextPage

Declaration: procedure NextPage;

NextPage procedure triggers the subclassed tNoteBook to switch to the next page.

Method PriorPage

Declaration: procedure PriorPage;

PriorPage procedure triggers the subclassed tNoteBook to switch to the previos page.

Method SetPage

Declaration: procedure SetPage(NewPageIndex: Integer; Next: Boolean);

Set the current page to the new page.

Class TPageProxy

PageProxy defin

Property PageManager

Declaration: PageManager: TPageManager;

PageManager contains the tNotebook custom control on the form.

Property PageName

Declaration: PageName: string;

PageName property is the name of the current notebook page that is active or shown.

Event OnEnter

Declaration: OnEnter: TPageNotifyEvent;

OnHide event is triggered when focus comes into the Notebook.

Event OnHide

Declaration: OnHide: TPageNotifyEvent;

OnHide event is triggered when a hidden page is triggered or accessed.

Example:

You want to password protect the form. Use OnHide to prevent the user from hacking the EXE file to trigger the active page instead <g>.

Event OnLeave

Declaration: OnLeave: TPageNotifyEvent;

OnLeave event is triggered when focus from a page from tNoteBook is no longer active.

Event OnShow

Declaration: OnShow: TPageNotifyEvent;

OnShow event is triggered when a page from tNoteBook is switched.

Type TPageNotifyEvent

Declaration: TPageNotifyEvent = procedure(Next: Boolean) of object;

TPageNotifyEvent is used for events of the TPageProxy class, which occurs when the notebook's page changed.

Type TPageOwner

Declaration: TPageOwner =TNotebook;

Defines type of PageOwner property of TPageManager component. Currently only TNotebook is supported as page owner.

Type TPageRequestEvent

Declaration: TPageRequestEvent = procedure(CurrentPage: Integer; var NewPage: Integer) of object;

TPageRequestEvent type is used for the OnGetNextPage and OnGetPriorPage events of the TPageManager component.

UNIT PageMngr Property NextBtn NextEnabled PageCount PageIndex PageIndex: Integer] PageOwner PageProxies PriorBtn PriorEnabled SetStartPage

Event OnCheckButtons OnGetNextPage OnGetPriorPage

Method GetNextPageIndex GetPriorPageIndex GetUniqueName NextPage PriorPage SetPage

Class TPageProxy

Property <u>PageManager</u> <u>PageName</u>

Event OnEnter OnHide OnLeave OnShow

Type TPageNotifyEvent TPageOwner TPageRequestEvent

Control Name	Unit	Class
Rx Calculator	<u>RxCalc</u>	TRxCalculator

Description:

Rx Calculator is a floating point calculator designed to do simple operations. It is an invisible component, or one that pops-up upon operation.

Use the <u>Execute</u> method to popup the Calculator. If any errors occur on calculators, use the <u>BeepsOnError</u> to control beeping on errors If you prefer the classical white calculator, set the <u>Clt3d</u> to false.

Method Execute

Method Execute starts the calculator.

Property BeepOnError

Set property BeepOnError to true and a beep will be generated.

BeepOnError determines whether a beep is sounded when an invalid character is entered. If BeepOnError is True, MessageBeep is called when an invalid character is entered or when a character is entered and the field is full.

Property Ctl3d

3D shaded gray or white.

Event OnCalcKey

OnCalcKey event is activated when the user presses any key.

Event OnChange

OnChange event is triggered when any value of the calculator is changed.

Event OnDisplayChange

The TRxCalculator component makes a simple Calculator dialog box available for your application, provides the four basic math functions (add, subtract, multiply, and divide), plus a few extras like memory, square root, percent, invert, and sign change.

Display the Calculator dialog box by calling the Execute method. The calculation result is stored in the Value property.

Property Memory Declaration: Memory: Double; Run-time and read-only. Memory contains the value of the calculator's memory register.

Property Precision

Declaration: Precision: Byte;

Precision determines the precision used in formatting the value in the Calculator display.

Use Precision set the number of digits beyond the decimal point the value should be formatted to before rounding begins.

The default value of Precision is 15 decimal places.

NOTE. Precision must be greater than or equal to 2. Trying to set Precision to a value less than 2 changes the value of Precision to 2.

Property Title

Declaration: Title: string;

The Title property determines the text that appears in the calculator dialog box's title bar.

In other words, the title of the Calculator

Property Value Declaration: Value: Double;

The Value property is used to retreive calculation results from the Calculator dialog box and to set current value in the calculator's display.

In other words, the value stored on the display.

Unit <u>RxCalc</u> Method <u>Execute</u>

Property BeepOnError Ctl3d Help Content

Event OnCalcKey OnChange OnDisplayChange

Property Memory Precision Title Value

Control Name	Unit	Class
RX String Holder	<u>StrHlder</u>	TStrHolder

Description:

TStrHolder is a TStrings wrapper component making it easier to work with the TStrings and TStringList objects at design time.

Using TStrHolder component you can set TStrings object properties and event handlers in Delphi form designer and hold a number of strings in your forms.

Property KeyString

Declaration: KeyString: string;

This is the string that is used to encode the contents of tStrHolder.

When this is empty, there is no encryption done. When this has a value, the string is used to encode the contents of the tStrHolder.

Note: The contents will then be stored in the DFM (Delphi Form) file as an encrypt data using a **simple XOR-alghoritm** with KeyString as the key to decode.

Duplicates : dupIgnore, dupAccept, dupError dupIgnore = Ignore (or delete) duplicates dupAccept = Accept duplicates dupError = Trigger an exception.

Property Sorted : boolean

Property Sorted ensures the string list is sorted when any changes occur.

Property String : tStringList

Property String is the main container to store the strings. See also: your Borland Help, tStringList

Event OnChange

Event OnChange is triggered when there is/are any changes to the list.

Event OnChanging

Event OnChanging is trigged during (after) changes to the list.

Procedure Clear;

Clear procedure cleares the string list.

Procedure Destroy; Destory rocedure destroys the String Holder.

StrHolderVariable := tStrHolder.Create

Constructor *Create* creates an instance of the tStrHolder

Unit <u>StrHlder</u> / String Holder

Procedures: Clear Destroy

Properties: Duplicates KeyString Sorted String tStrHolder.Create

Events: OnChange OnChanging

Control Name	Unit	Class
RX Application Events	<u>AppEvent</u>	TAppEvent

Description:

TAppEvent is a Application wrapper component, which makes it easier to work with the TApplication object properties and events at design time. By using TAppEvent component you can set TApplication properties and event handlers in Delphi form designer.

In other words:

Instead of writing code to respond to application event handlers. This is beneficial as instead of writing code all over the place, you can have a concise and disciplined manner to store the information.

Example: if you set the property <u>Chain</u> = True, Set the show-hint to true.

Property Chained

Declaration: Chained: Boolean;

The Chained property indicates whether this component should replace any other TAppEvents event handlers of the same type. When Chained is False, the specified event handler replaces any previously registered by TAppEvents objects event handlers of the same type. When Chained is True, the corresponding event handler is called in addition to other registered events (by other TAppEvents objects).

Event OnPaintIcon

Declaration: OnPaintIcon: TNotifyEvent;

OnPaintIcon occurs when the minimized application receives a Windows paint message. Use OnPaint to perform special processing when the application's icon is redrawn. Any special painting should be done in this event. You should use the Canvas property to paint application's icon.

Event OnSettingsChanged

Declaration: OnSettingsChanged: TNotifyEvent;

OnSettingsChanged event occurred when the user alters the Windows system configuration and application receives a WM_WININICHANGE message.

Property ShowHint : Boolean

Showhint determines whether the application should show hints or not.

Property HintHidePause : Boolean

HintHidePause determines whether the pause time before hiding the hint (or delay time after the mouse moves and the hint gets hidden).

Property HintColour : color

HintColour determines the colour of the hint.

Property HintShortPause : Longint

This variable determines the amount of time required before the hint shows up. (That is, you put the mouse on the form, how long it takes for the hint to come out after you put the mouse on the form).

Property ShowMainForm : Boolean

Show the main form be shown? Setting this variable to **true** means the form will be shown.

Property ShowHint : Boolean

ShowHint shows the hint directly without having to write any code.

Event OnActivate Method DoActivate(Sender: TObject);

OnActivate is triggered when the form is activated.

Event OnDeactivate Method DoDeactivate(Sender: TObject);

OnDeactivate is triggered when the form is deactivated.

Event OnException Method DoException(Sender: TObject; E: Exception);

OnEception is triggered when there is an error (or exception on the form).

Event OnIdle Method DoIdle(Sender: TObject; var Done: Boolean);

OnIdle is triggered when the application is idle.

Event OnHelp

function DoHelp(Command: Word; Data: Longint; var CallHelp: Boolean): Boolean;

OnHelp is triggered when help is triggered (useful for putting all the code inside your code instead of triggering it all over the place).

Event OnHint

Method DoHint(Sender: TObject);

DoHint is triggered when hint is triggered.

Event OnMessage

Method DoMessage(var Msg: TMsg; var Handled: Boolean);

Event OnMessage is triggered when hint is triggered.

Event OnMinimize

Method DoMinimize(Sender: TObject);

OnMinimize is triggered when your application is minimized..

Event OnRestore

Method DoRestore(Sender: TObject);

OnRestore is triggered when your application is restored (from maximized position, or from minimized position).

Event OnShowHint

Method DoShowHint(var HintStr: string; var CanShow: Boolean; var HintInfo: THintInfo);

OnShowHint is triggered when a hint is shown.

Unit <u>Application Events</u> Properties: <u>Chained</u> <u>Hint Colour</u> <u>HintHidePause</u> <u>HintShortPause</u> <u>ShowHint</u> <u>ShowMainForm</u> Events: <u>OnPaintIcon</u>

OnSettingsChanged OnActivate OnDeactivate OnHelp OnHint OnIdle OnMessage OnMinimize OnRestore OnPaintIcon OnSettingsChanged

Control Name	Unit	Class
RX Timer List	<u>TimerLst</u>	TRxTimerEvent

Description:

This component intend for storage of the event of TRxTimerList events queue. TRxTimerEvent have the same properties as TTimer: Enabled, Interval, OnTimer. Additional event may be "single" - stand inactive after one execute of OnTimer (it action depend on Cycled property value).

Class TRxTimerList

The TRxTimerList component provides all the functions of the standard TTimer component, plus the additional benefit of using only one Windows timer for up to 32767 timing events. The practical maximum number of timing events depends on how long it takes your application to service each timer notification.

Windows (system) timers are a scarce system resource, especially if 16-bit application. Using more system timers than necessary can impact performance. If a (system) timer is not available when a program needs one, it can either terminate gracefully or it can terminate in a manner that leaves Windows in an unstable state. You can reduce the chances of running out of timers by using a TRxTimerList rather than multiple timers.

TRxTimerList uses the concept of a timer events to perform the same function as the TTimer's OnTimer event. After adding a TRxTimerList component to your form, use component editor to create events at design-time, or call <u>Add</u> method at run-time to create timer events dinamically. Events can repeat until they are canceled or they can be fired only once and then automatically removed (accordingly Cycled property value).

Property Active

Declaration: Active: Boolean;

The Active property controls whether the timer's *should* respond to timer events.

Property Count

Declaration: Count: Integer;

Count returns the total number of timer events. The number returned by Count includes both enabled and disabled timers.

Property EnabledCount

Declaration: EnabledCount: Integer;

EnabledCount returns the number of enabled timer events (which Enabled property is True).

Property Events Declaration: Events: TList; List included pointers to events defined by class TRxTimerEvent. Property intend for internal use. In components work time events order in this list may be changed. You can use for access to events data method ItemByHandle and for dynamic add a new events method Add.

Event OnFinish

Declaration: OnFinish: TNotifyEvent;

This is triggered when an event has been completed.

Event OnTimers

Declaration: OnTimers: TAllTimersEvent;

The OnTimers defines an event handler that is called for each timer events.

Method Activate

Declaration: procedure Activate;

Activate and Deactivate methods controls whether the timer's responds to timer events. Call Activate to set the Active property for the timer list to True.

Method Add

Declaration: function Add(AOnTimer: TNotifyEvent; AInterval: Longint; ACycled: Boolean): Longint;

You can use this method to add new event.

Parameters AOnTimer, AInterval and ACycled contained values of properties OnTimer, Interval

and Cycled accordingly for new event. Handle of this event returned as method result.

Method Clear

Declaration: procedure Clear; This method removes all events from the list.

Method Deactivate

Declaration: procedure Deactivate;

Activate and Deactivate methods controls whether the timer's responds to timer events. Call Deactivate to set the Active property for the timer list to False.

Method Delete

Declaration:

procedure Delete(AHandle: Longint);

Method removes event specified by Handle from the list.

Method ItemByHandle

Declaration: function ItemByHandle(AHandle: Longint): TRxTimerEvent;

This returns the handle to the event by ordinal number.

In Simple English: This converts a "handle to the event" to the actual event.

Type TAllTimersEvent

Declaration: TAllTimersEvent = procedure(Sender: TObject; Handle: Longint) of object;

TAllTimersEvent is the type of the OnTimers event of the TRxTimerList component.

Property AsSeconds

Declaration: AsSeconds: Word;

This is conversion property. AsSecond can be used to read or set the value of Interval property in seconds.

Property Cycled

Declaration: Cycled: Boolean;

Set the Active property to True to define what event to occur whenever a specified period of time passes. You use the Interval property to control the amount of time between timer events (in milliseconds). You can not use RepeatCount property anyhow. Set the Active property to False to define what event to occur only one time and after it Enabled property has been False value.

You can use RepeatCount to define number of occured times. You can use ExecCount

property for definition of the times occured events after execute TRxTimerList.

Property ExecCount

Declaration: ExecCount: Integer;

Run-time and read only. Property stored number of occured times after execute Activate method of TRxTimerList. When Active property of TRxTimerList set in False then value of ExecCount is equal to zero.

Property Handle

Declaration: Handle: Integer;

Run-time and read only. The Handle property provides access to the event by unique identificator.

Property RepeatCount

Declaration: RepeatCount: Integer;

Property stored number of times what need occur event whenever a specified period of time passes after execute Activate method of TRxTimerList. After it property Enabled has been False value.

Value of this property must be more greater or equal 1. May be used only when Cycled property value is False.

You can use ExecCount property for definition of the times occured events after execute TRxTimerList.

Property TimerList

Declaration: TimerList: TRxTimerList;

This contains a "List of events"

Unit <u>Timer List</u> **Properties: Active Events Procedures:** Count **EnabledCount** <u>Activate</u> <u>Add</u> <u>Clear</u> **Deactivate Delete TAllTimersEvent AsSeconds Cycled ExecCount Handle RepeatCount** TimerList

Events OnFinish OnTimers

Control Name	Unit	Class

RX MRU [Most Recently Used] List or Manager

TMRUManager

Description:

This component greatly simplifies adding MRU (More Recently Used) object lists (i.e. file descriptions) to the application menus.

Example Usage:

It is somewhat difficult to explain what the MRU manager does, and is best explained by an example:

Under the File menu, I have a sub-menu called "Recent". So, under the RecentMenu property of the MRUManager, I have this menu item. I save up to 10 recently used files in the registry. The call I use to load these files from the registry is:

MRUManager.LoadFromRegistry(EMIni.CSRegistry,'Recent');

where EMIni.CSRegistry is the registry key and recent is the section. Check out the Rx code in the MRUList.PAS unit for any more details on this. To save a file to the MRUManager, I use:

MRUManager.Add(OpenDialog.Filename,0)

Where OpenDialog.Filename is the name of the file to be added to the list and 0 is a UserData flag which I don't use. Again, the MRUList.PAS has the source for this.

Then, when I close my program, I save the info which was added to the MRUManager by calling:

MRUManager.SaveToRegistry(EMIni.CSRegistry,'Recent');

Property AccelDelimiter

Declaration: AccelDelimiter: TAccelDelimiter;

This defines the accelerated delimited.

In other words: Since "&" cannot be stored in, it is substituted to either a tab or space instead. The property AccelDelimiter sets this.

Property AutoEnable

Declaration: AutoEnable: Boolean;

AutoEnable property automatically enables the MRU manager on startup and shutdown of the form. In other words, all the processing is done automatically for you.

Property AutoUpdate

Declaration: AutoUpdate: Boolean;

Set AutoUpdate to true, the MRU is automatically updated.

In other words, the MRU manager automatically processes and updates itself automatically when AutoUpdate property is true.

Property Capacity

Declaration: Capacity: Integer;

Capacity is the maximum number of strings the MRU list can hold. Adding new strings will delete oldest strings if necessary.

Property Mode

Declaration: Mode: TRecentMode;

Property mode controls how the entries are added into the system. They are either:

- mrInsert inserted into the most recently used list.
- mrAppend appended into the most recently used list.

Property RecentMenu

Declaration: RecentMenu: TMenuItem;

RecentMenu is the last or recent menu that would be utilised.

Property RemoveOnSelect

Declaration: RemoveOnSelect: Boolean;

RemoveOnSelect defines whether the most recently used entry is deleted when the entry is selected.

Property SeparateSize

Declaration: SeparateSize: Word;

SeparateSize - separate size is the MRU manager's stuff.

Property ShowAccelChar

Declaration: ShowAccelChar: Boolean;

ShowAccelChar property defines whether the Accelerated

Property StartAccel

Declaration: StartAccel: Cardinal;

StartAccel defines if the each member of the MRU list should start with an accelerated character.

Property Strings

Declaration: Strings: TStrings;

Strings property can be used to direst access to list of elements stored in the TMRUManager component. By using this property you can assign MRU-strings to another controls, for example, to list-boxes or to comboboxes:

ListBox1.Items := MRUManager1.Strings;

Event OnChange

Declaration: OnChange: TNotifyEvent;

OnChange occurs immediately after the list of MRU-strings changes. Write an OnChange event handler to respond to changes in the list of MRU-elements. For example, if you are using MRU-strings in a control (list-box, combo-box or another), the OnChange event handler could be used to change string list associated with a control.

Event OnClick

Declaration: OnClick: TClickMenuEvent;

OnClick event is triggered when the menu item in the MRU list is clicked.

Event OnGetItemData

Declaration: OnGetItemData: TGetItemEvent;

OnGetItemData event is triggered when an item is retrieved from the MRU list.

Event OnReadItem

Declaration: OnReadItem: TReadItemEvent;

OnReadItem event is triggered when an item or data is read into the MRU list.

Event OnWriteItem

Declaration: OnWriteItem: TWriteItemEvent;

OnWriteItem event is triggered when an item or data is written into the MRU list.

Method Add

Declaration: procedure Add(const RecentName: string; UserData: Longint);

procedure Add adds an entry RecentName to the MRU list and UserData.

Method LoadFromIni

Declaration: procedure LoadFromIni(Ini: TIniFile; const Section: string);

LoadFromIni loads the MRU list from the INI file.

Method LoadFromRegistry

Declaration: procedure LoadFromRegistry(Ini: TRegIniFile; const Section: string);

LoadsFromRegistry loads the MRU list from the registry.

Method Remove

Declaration: procedure Remove(const RecentName: string);

Remove a single item from the list. No error occured when item not found.

Method SaveToIni

Declaration: procedure SaveToIni(Ini: TIniFile; const Section: string);

SaveToIni procedure saves the MRU list to an INI file instead.

Method SaveToRegistry

Declaration: procedure SaveToRegistry(Ini: TRegIniFile; const Section: string);

SaveToRegistry saves the MRU list is the registry.

Method UpdateRecentMenu

Declaration: procedure UpdateRecentMenu;

UpdateRecentMenu updates the MRU list. Use this if the <u>autoupdate property</u> is false and you want to update the MRU list manually instead.

Type TAccelDelimiter

Declaration: TAccelDelimiter = (adTab, adSpace);

TAccelDelimiter is a set of values for the AccelDelimiter property of the TMRUManager component.

Type TClickMenuEvent

Declaration: TClickMenuEvent = procedure (Sender: TObject; const RecentName, Caption: string; UserData: Longint) of object;

TClickMenuEvent is the type of the OnClick event of the TMRUManager component.

Type TGetItemEvent

Declaration: TGetItemEvent = procedure (Sender: TObject; var Caption: string; var ShortCut: TShortCut; UserData: Longint) of object;

TGetItemEvent is the type of the OnGetItemData event of the TMRUManager component.

Type TReadItemEvent

Declaration: TReadItemEvent = procedure (Sender: TObject; IniFile: TObject; const Section: string; Index: Integer; var RecentName: string; var UserData: Longint) of object;

TReadItemEvent is the type of the OnReadItem event of the TMRUManager component.

Type TRecentMode

Declaration: TRecentMode = (rmInsert, rmAppend);

TRecentMode is a set of values for the Mode property of the TMRUManager component.

Type TWriteItemEvent

Declaration: TWriteItemEvent = procedure (Sender: TObject; IniFile: TObject; const Section: string; Index: Integer; const RecentName: string; UserData: Longint) of object;

TWriteItemEvent is the type of the OnWriteItem event of the TMRUManager component.

Unit <u>MRUList</u> Property <u>AccelDelimiter</u> <u>AutoEnable</u> <u>AutoUpdate</u> <u>Capacity</u> <u>Mode</u> <u>RecentMenu</u> <u>RemoveOnSelect</u> <u>SeparateSize</u> <u>ShowAccelChar</u> <u>StartAccel</u> <u>Strings</u>

Events OnChange OnClick OnGetItemData OnReadItem OnWriteItem

Method Add

LoadFromIni LoadFromRegistry Remove SaveToIni SaveToRegistry UpdateRecentMenu

Type TAccelDelimiter TGetItemEvent TReadItemEvent TRecentMode TWriteItemEvent

Control Name	Unit	Class
Tray Icon	RxShell	TRxTrayIcon

Description:

The TRxTrayIcon component enables 32-bit Delphi applications to add icons to the Windows 95 or Windows NT 4.x tray. The icon can be added to the tray by setting the <u>Active</u> property to true, and can be removed from the tray by setting the <u>Active</u> property to false.

The <u>Icon</u> property specifies static icon to the tray. If you want animate icons in the tray, set values to the Icons property and set the <u>Animated</u> property to True. You can test tray-icon in design-time by setting the <u>ShowDesign</u> property to True. To respond to mouse events for the icon, use the <u>OnClick</u> and OnDblClick events.

Note:

To display a pop-up menu when the user clicks the right mouse button on the Tray Icon, assign a PopupMenu component to the PopupMenu property and set the AutoPopup property of the menu to True.

To display a Help Hint whenever the mouse pointer pauses over the icon momentarily, set the Hint property to a help string.

You can use 16x16 icons with the TRxTrayIcon component as well as 32x32 icons.

Property Active

Declaration: Active: Boolean;

Set the Active property to True to add an Icon to the Windows 95 Tray. Set the Active property to False to remove the Icon from the Windows 95 Tray.

If the <u>Animated</u> property is True, the Icons property used to show animation in the Tray, if the <u>Animated</u> property is False, then icon specified by the Icon property used to show static icon in the Windows 95 Tray.

Property Animated

Declaration: Animated: Boolean;

The Animated property is used to show animated icons in Windows 95 Tray. The icons to be used in the animation are listed in the Icons property, and the time to wait between animation frames is specified by the Interval property.

Property Icons

Declaration: Icons: TIconList;

The Icons property is used to define the animation that will occur when a Tray Icon has <u>Active</u> property set to True and <u>Animated</u> property set to True also (if Animated property is False, the static Icon property use instead). The time to wait between animation frames is specified by the Interval property. The property editor of the Icons property allows you to load Windows animation cursors (ani-files) to this property.

Property ShowDesign

Declaration: ShowDesign: Boolean;

This property allows you to testing tray icon in design-time. (This property is Ignored during run-time.)

Event OnClick

Declaration: OnClick: TMouseEvent;

To respond to mouse click events (left or right button) for the icon in Windows System Tray, use an OnClick event.

Type TExecState

Declaration: TExecState = (esNormal, esMinimized, esMaximized, esHidden);

TExecState type defines the four possible initial states of the launched applications main window for the InitialState parameter of FileExecute and FileExecuteWait functions: normal, minimized, maximized or hidden.

Routine FileExecute

Declaration: function FileExecute(const FileName, Params, StartDir: string; InitialState: TExecState): THandle;

The FileExecute function opens a specified file. The file can be an executable file or a document file. If FileName specifies an executable file, Params is a string that specifies parameters to be passed to the application. StartDir specifies the default directory. If FileName specifies an executable file, InitialState specifies how the application is to be shown

when it is opened. This parameter can be one of the following values: esNormal, esMinimized, esMaximized or esHide. If the function succeeds, the return value is the instance handle of the application that was run. If the function fails, the return value is an error value that is less than or equal to 32.

FileExecute example:

if FileExecute('NOTEPAD.EXE', ", 'C:\', esNormal) <= 32 then
ShowMessage('Can't start NOTEPAD.EXE');</pre>

Routine FileExecuteWait

Declaration: function FileExecuteWait(const FileName, Params, StartDir: string; InitialState: TExecState): Integer;

This function is similar to the FileExecute function, but it don't return until after a launched program finishes executing.

32-bit version: If the function succeeds, the return value is termination status of the executed process. If the

function fails, the return value is -1.

16-bit version: If the function succeeds, the return value is zero. If the function fails, the return value is -1.

> FileExecuteWait example:

if not FileExecuteWait('NOTEPAD.EXE', 'readme.txt', 'C:\', esNormal) then ShowMessage('Can not start NOTEPAD.EXE');

Routine IconExtract

Declaration: function IconExtract(const FileName: string; Id: Integer): TIcon;

The IconExtract function creates the TIcon object from the given executable file, dynamic-link library (DLL), or icon file. The FileName parameter specifies the name of an executable file, DLL, or icon file. The ID parameter specifies the index of the icon to retrieve. If this value is 0, the function returns the handle of the first icon in the specified file. If the function succeeds, the return value is the pointer to the Icon-object. After using the icon you must destroy it by Free method.

IconExtract example:

```
var
  Ico: TIcon;
begin
  Ico := IconExtract('PROGMAN.EXE', 0);
  try
    { use icon }
  finally
    Ico.Free;
  end;
end;
```

Routine WinAbout

Declaration: procedure WinAbout(const AppName, Stuff: string);

The WinAbout procedure displays a Shell About dialog box.

Please note that the WinAbout procedure dialog box uses text and a default icon that are specific to either Microsoft Windows or Microsoft Windows NT. An example of a Shell About dialog box can be seen by selecting the About Program Manager command in Program Manager.

The AppName parameter is the text that the procedure displays in the title bar of the About dialog box and on the first line of the dialog box after the text "Microsoft Windows" or "Microsoft Windows NT". If the text contains a "#" separator, dividing it into two parts, the function displays the first part in the title bar, and the second part on the first line after the text "Microsoft Windows" or

"Microsoft Windows NT".
The Stuff parameter is the text that the function displays in the dialog box after the version and copyright information.

> WinAbout example:WinAbout('Delphi Demo Application', 'Borland Int., 1995');

Unit <u>RX Shell</u> Property <u>Active</u> <u>Animated</u> <u>IconExtract</u> <u>Icons</u> <u>ShowDesign</u>

Events: OnClick

Types: <u>TExecState</u>

Methods <u>WinAbout</u> <u>FileExecute</u> <u>FileExecuteWait</u>

Control Name	Unit	Class
Icon List	IcoList	TIconList

Description: The TIconList class is a container for a group of icons. This class used internally by <u>TRxTrayIcon</u> component. To add an icon to the icon list, use the <u>Add</u>, <u>AddResource</u>, <u>Insert</u>, <u>InsertResource</u> and <u>LoadResource</u> methods. To retrieve an icons by its index from the icon list, use the <u>Icons</u> property and the <u>Indexof</u> property to obtain the icon listing from it's index.

Property Icons[Index: Integer]

Declaration: Icons[Index: Integer]: TIcon;

The Icons property returns the icon specified by Index as an TIcon object.

Method Add

Declaration: function Add(Icon: TIcon): Integer; virtual;

The Add method adds an icon to the list of icons. Specify the icon to insert as the value of the Icon parameter. Add returns the index of the added icon.

Method AddResource

Declaration: function AddResource(Instance: THandle; ResId: PChar): Integer; virtual;

The AddResource method loads an icon resource by ResID identifier and adds its to the list of icons. AddResource returns the index of the added icon.

Method IndexOf

Declaration: function IndexOf(Icon: TIcon): Integer; virtual;

The IndexOf method returns the position of an icon in a list. The first position in a list is 0. If an icon is not in the list, IndexOf returns -1.

Method Insert

Declaration: procedure Insert(Index: Integer; Icon: TIcon); virtual;

The Insert method inserts an icon into the list of icons. Specify the icon to insert as the value of the Icon parameter. Specify the position in the list where you want the icon inserted as the value of the Index parameter. The index is zero-based, so the first position in the list has an index value of 0.

Method InsertResource

Declaration: procedure InsertResource(Index: Integer; Instance: THandle; ResId: PChar); virtual;

The InsertResource method loads an icon resource by ResID identifier and inserts its into the list of icons. Specify the position in the list where you want the icon inserted as the value of the Index parameter.

Method LoadResource

Declaration: procedure LoadResource(Instance: THandle; const ResIds: array of PChar);

The LoadResource method loads the icons resources by ResIDs identifiers and adds them to the list of icons.

Unit <u>IcoList</u>

Method Add AddResource IndexOf Insert InsertResource LoadResource Property Icons[Index: Integer]

Control Name	Unit	Class
RX ANI File.	<u>AniFile</u>	TAnimatedCursorImage

Description:

The TAnimatedCursorImage object encapsulates the data loaded from a Windows animated cursor (.ANI) file.

(An ANI file is a file that contains a list of icons that animate during run-time)

To load an ANI cursor image from a file, call the <u>LoadFromFile</u> method. To retrieve a single image as an icon by its index from the list after loading, use the <u>Icons</u> and <u>IconCount</u> properties.

You can also assign the images from the animated cursors to the bitmap that consists of a series of frames arranged one after other by using the <u>AssignToBitmap</u> method.

To extract the copyright information, utilise the property <u>Creator</u> and the <u>title</u> property to obtain the description of what the ANI file is about. To control the speed of the rate of animation, use the <u>default rate</u> property. If your background is of some other colour or you would like your ANI file to be background transparent, use the <u>original colours</u> property to tweak the background colour.

See also:

RX Animate (TAnimatedImage) and RX Gif Animator (TRxGIFAnimator).

Property Creator

Declaration: Creator: string;

After loading ANI cursor file into the TAnimatedCursorImage object, the Creator property contains the information about the author that produced the file or some another copyright information.

Property DefaultRate

Declaration: DefaultRate: Longint;

The DefaultRate property determines the amount of time in milliseconds that passes among frames of a TAnimatedCursorImage animation. This value is read from .ANI file and is equal to the duration of first cursor's image.

Property IconCount

Declaration: IconCount: Integer;

IconCount is read only and contains the number of icons in a list after loading ANI data from a ANI Cursor file.

Property Icons[Index: Integer] Declaration: Icons[Index: Integer]: TIcon;

The Icons property allows you to access a specific icon inside the icon list after loading the data from the .ANI file.

Property OriginalColors

Declaration: OriginalColors: Word;

Read only. The OriginalColors determines in what bit format (color count) the cursor images are stored in the animated cursor file.

The OriginalColors will always be one of the specified value: 2 (monochrome icons), 16 (16-colors icons), 256 (256-colors icons) or 0 (true-color or high-color icons).

Property Title

Declaration: Title: string;

The Title property stores the description of the ANI file.

Method AssignToBitmap

Declaration: procedure AssignToBitmap(Bitmap: TBitmap; BackColor: TColor; DecreaseColors, Vertical: Boolean);

The procedure AssignToBitmap discards any current image in Bitmap object and replaces the icon's images loaded from ANI-file. The resulting image is a series of frames arranged one after the other according to the Vertical parameter value. To make resulted image partially transparent you could use the <u>BackColor</u> parameter. Also you could use DecreaseColors property to decrease bitmap color count to the original value described in the ANI-file.

Method LoadFromFile

Declaration: procedure LoadFromFile(const Filename: string); virtual;

The LoadFromFile method reads the animated cursor file specified in FileName and loads the icons into the TAnimatedCursorImage object.

Unit <u>AniFile</u>

Property <u>Creator</u> Property <u>DefaultRate</u> Property <u>Icons Count</u> Property <u>Icons</u> Property <u>OriginalColours</u> Property <u>Title</u>

Method <u>LoadFromFile</u> Method <u>AssignToBitmap</u>

Control Name	Unit	Class
RX ANI File.	<u>Animate</u>	TAnimatedImage

Description: The TAnimatedImage component can display simple animation sequences on the form on which it is placed.

Use the TAnimatedImage component to display a sequence of frames with a set interval to create an animated image. You specify the bitmap containing the frames in the <u>Glyph</u> property. Set the <u>Active</u> property to True and watch the animation.

You can use design-time component editor to load <u>Windows animated cursors (ANI files</u>) into the <u>Glyph</u> property.

The animation sequence is stored in the <u>Glyph</u> property. The bitmap may consist of a series of frames arranged one after the other (from left to right or from top to bottom, according to <u>Orientation</u> property value), while the <u>Interval</u> property controls the rate at which the frames are displayed. The <u>NumGlyphs</u> property specifies count of the frames in the image.

It is possible to define number of frames to be displayed in the present moment, during runtime based on the value of the property <u>GlyphNum</u>. You can also set this value during design time.

The property <u>Opaque</u> determines, whether the component will fill in a background under image by colour Color or it will be completely transparent. Component also includes properties similar to the properties TImage. You can also use a component TAnimatedImage for a drawing on the form of the transparent motionless images, for example, for overlay of the images against each other.

The <u>TransparentColor</u> property sets the color for which the form's color should be displayed instead of the bitmap color. When you set the <u>Active</u> property to True, the glyph starts to animate. The <u>OnStart</u> event is triggered when the animation started and the <u>OnStop</u> event is generated when the property <u>Active</u> sets to False.

To display the frame sequence at timed intervals (or in other words, animate the glyph):

- Set the <u>Glyph</u> property to the frame sequence.
- Set the <u>NumGlyphs</u> property desired frame numbers in the image.
- Set the <u>TransparentColor</u> and <u>Opaque</u> properties to control image transparency.
- Set the <u>Interval property</u> to the desired time interval between successive frames.
- Set the <u>Active</u> property to True.
- You can stop the animation sequence by setting the <u>Active</u> property to False.

See also:

RX ANI file (TAnimatedCursorImage) and RX Gif Animator (TRxGIFAnimator)

Property Active Declaration: Active: Boolean; Once the frame sequence is properly loaded in the <u>Glyph</u> variable and the properties <u>NumGlyphs</u>, <u>GlyphNum</u>, <u>InactiveGlyph</u> and <u>Interval</u> properties are set, setting **Active** property to **True** will cause the animation to start (and stop when the Active property is **False**).

Property Glyph

Declaration: Glyph: TBitmap;

The **Glyph** property holds the animation sequence frames . The frames should be in Windows Bitmap file format. The frames may be arranged in either vertical or horizontal fashion one after the other.

Property GlyphNum

Declaration: GlyphNum: Integer;

The **GlyphNum** property stores the frame number of the frame currently displayed. It can be set to the frame number which needs to be displayed while <u>Active</u> property is False.

Note: When the <u>InactiveGlyph</u> property is not equal to -1, the <u>GlyphNum</u> property value are ignored.

When setting this property, if the value is within the acceptable range (between 0 and the number of frames in the Glyph) the current frame display is changed.

Note: Before using this property, ensure that the animation sequence if properly loaded into the <u>Glyph</u> property.

Property InactiveGlyph

Declaration: InactiveGlyph: Integer;

The **InactiveGlyph** property specifies the frame number of the frame displayed when <u>Active</u> is set to False. When the InactiveGlyph property value is equal to **-1**, the <u>GlyphNum</u> property can be set to the frame number which needs to be displayed. When **InactiveGlyph** is within the acceptable range (between 0 and the number of frames in the bitmap) then this value used to display the image in inactive (non-animated) state and this frame not used in during animation (i.e. when <u>Active</u> = True).

Property Interval

Declaration: Interval: Word;

The **Interval** property specifies in milliseconds the amount of time required between changing the frames of a <u>TRxTrayIcon</u> (32-bit only), <u>TAnimatedImage</u> or <u>TRxDice</u> animation; or speed of text scrolling in the <u>TSecretPanel</u> component.

Property NumGlyphs

Declaration: NumGlyphs: Integer;

The **NumGlyphs** property specifies the number of frames in the image contained in the <u>Glyph</u> property.

Property Opaque

Declaration: Opaque: Boolean;

Opaque specifies whether the background of the image obscures objects below the animated image object.

- Set **Opaque** to False to allow objects behind the TAnimatedImage object to show through the background of the bitmap.
- Set **Opaque** to True to make the background of the bitmap opaque. Opaque has no effect unless the <u>TransparentColor</u> property not equal to clNone.

Property Orientation

Declaration: Orientation: TGlyphOrientation;

Orientation determines whether the frames are placed horizontal or vertical fashion in the image specified by <u>Glyph</u> property of the TAnimatedImage component. It can either be goHorizontal or goVertical. The default value is **goHorizontal**.

Property TransparentColor

Declaration: TransparentColor: TColor;

TransparentColor determines which color of the bitmap is to be transparent when the frame is drawn. This property makes the frame partially transparent by setting the color in the bitmap which should be replaced by the parent's background (when <u>Opaque</u> = False) or by the Color property value (when <u>Opaque</u> = True).

Event OnStart

Declaration: OnStart: TNotifyEvent;

The OnStart event is triggered if the component has started displaying the animation and the <u>Active</u> (Animate) property is set to True.

Event OnStop

Declaration: OnStop: TNotifyEvent;

The OnStop event is triggered if the component has finished displaying the animation and the <u>Active</u> (Animate) property is set to False (during run-time).

Type TGlyphOrientation

Declaration: TGlyphOrientation = (goHorizontal, goVertical);

This type defines possible values for **Orientation** property of a TAnimatedImage component. The value

goHorizontal sets horizontal (from left to right) alternation of frames and the value *goVertical* - vertical (from top to bottom).

Unit <u>Animate</u> Property: Active Glyph GlyphNum InactiveGlyph Interval NumGlyphs Opaque Orientation TGlyphOrientation

Events: OnStart OnStop

Type: <u>TransparentColor</u>

Control Name	Unit	Class
RX ANI File.	Parsing	TRxMathParser

Description:

The TRxMathParser is a class for parsing and evaluating simple mathematical expressions specified at runtime. The programming interface is simple: specify expression to be evaluated in AFormula parameter of Exec method, call Exec and retrieve computed value as result of Exec method.

When calculating error occured the ERxParserError exception is raised.

Accepted operators: + , - , * , /

The following functions are supported; it doesn't matter if you use lower or upper case: Arctan, Cos, Sin, Tan, Abs, Exp, Ln, Log, Sqrt, Sqr, Int, Fraq.

Method Exec

Declaration: function Exec(AFormula: String): Double;

Call Exec to retrieve computed value of expression specified by AFormula parameter. When calculating error occured the ERxParserError exception is raised.

Method Exec example

if rxMathParser1.Exec('2+Cos(2)') then Label1.Caption := 'Ok! Result = ' + FloatToStr(rxMathParser1.ResValue);

Routine GetFormulaValue

Declaration: function GetFormulaValue(const Formula: string): Double;

Call GetFormulaValue to retrieve computed value of expression specified by Formula parameter. This function uses TRxMathParser class to calculate an expression.

GetFormulaValue example:

Value := GetFormulaValue('2 * Sqrt(4)');
{ here Value is 4 }

Unit <u>Mathematical Parser</u> Property <u>Exec</u> <u>GetFormulaValue</u>

Control Name	Unit	Class
Version Information	<u>VerInfo</u>	TVersionInfo

Description:

This manipulates the information from the Microsoft Version Resource (VERSIONINFO) structure. Whenever the **FileName** property is changed, the class properties are updated. The read-only **Valid** property determines whether a version resource loaded successfully and information is valid. For more information about version resources, see your Windows API reference.

Property Comments

Declaration: Comments: string;

This contains the string specifying the comments..

Property CompanyName

Declaration: CompanyName: string;

This contains the string specifying the company that produced the file.

Property FileDescription

Declaration: FileDescription: string;

This contains the string specifying the file description of the file. This string may be displayed in a list box when the user is choosing files to install.

Property FileName

Declaration: FileName: string;

This contains the string specifying the FileName property where to search for Version Resource information. When the filename changes, the class looks for new version information and updates the class properties.

Property FileVersion

Declaration: FileVersion: string;

This contains the string specifying the version number of the file--for example, "3.10" or "5.00.RC2". For more information about version, see the Windows API reference.

Property FixedFileInfo

Declaration: FixedFileInfo: PVSFixedFileInfo;

This data structure contains a VSFixedFileInfo structure that specifies arbitrary data associated with this structure:

TVSFixedFileInfo = record

dwSignature: Longint; dwStrucVersion: Longint; dwFileVersionMS: Longint; dwFileVersionLS: Longint; dwProductVersionLS: Longint; dwProductVersionLS: Longint; dwFileFlagsMask: Longint; dwFileFlags: Longint; dwFileOS: Longint; dwFileOS: Longint; dwFileESubtype: Longint; dwFileDateMS: Longint; dwFileDateLS: Longint; end;

Property InternalName

Declaration: InternalName: string;

This contains the string specifying the internal name of the file, (if exists)

For example, a module name if the file is a dynamic-link library. If the file has no internal name, this string should be the original filename, without extension.

Property LegalCopyright

Declaration: LegalCopyright: string;

This contains the string specifying copyright notices that apply to the file. This should (or would) include the full text of all notices, legal symbols, copyright dates, and so on.

Property LegalTrademarks

Declaration: LegalTrademarks: string;

This contains the string specifying all trademarks and registered trademarks that apply to the file. This should include the full text of all notices, legal symbols, trademark numbers, and so on.

Property OriginalFilename

Declaration: OriginalFilename: string;

This contains the string specifying the original name of the file (do not including the path!). This information enables an application to determine whether a file has been renamed by a user. The format of the name depends on the file system for which the file was created.

Property PrivateBuild

Declaration: PrivateBuild: string;

This contains the string specifying information about a private version of the file--for example, "Built by TESTER1 on \TESTBED". This string should be present only if the VS_FF_PRIVATEBUILD flag is set in the dwFileFlags member of the FixedFileInfo property (VS_FIXEDFILEINFO structure) of the version resource.

Property ProductName

Declaration: ProductName: string;

This contains the string specifying the name of the product with which the file is distributed.

Property ProductVersion

Declaration: ProductVersion: string;

This contains the string specifying the version of the product with which the file is distributed--for example, "3.10" or "5.00.RC2". For more information about version, see the Windows API reference.

Property SpecialBuild

Declaration: SpecialBuild: string;

This contains the string specifying how this version of the file differs from the standard version--for example, "Private build for TESTER1 solving mouse problems on M250 and M250E computers". This string should be present only if the VS_FF_SPECIALBUILD flag is set in the dwFileFlags member of the FixedFileInfo property (VS_FIXEDFILEINFO structure) of the version resource.

Property Translation

Declaration: Translation: Pointer;

This contains the string specifying the translation table in the variable information structure. This property retrieves a pointer to an array of language and character-set identifiers.

Property Valid

Declaration: Valid: Boolean;

The read-only Valid property determines whether a version resource loaded successfully and information is valid. You can check this property after creation a resource object and after the FileName property has been changed.

Property VersionCharSet

Declaration: VersionCharSet: TVersionCharSet;

Specifies one of the character-set identifiers.

Property VersionLanguage

Declaration: VersionLanguage: TVersionLanguage;

Specifies one of the language identifiers.

Property VersionNum

Declaration: VersionNum: Longint;

Specifies the binary version number for the file.

Method Create

Declaration: constructor Create(const AFileName: string);

The Create method allocates memory to create a TVersionInfo object, passes it the file name of the interesting file and read version information about a specified file from VERSIONINFO resource. If the reading succeeds, the Valid property is True. If the reading fails, the Valid property is False.

Method GetVerValue

Declaration: function GetVerValue(const VerName: string): string;

The GetVerValue function returns selected version information from the specified version-information resource.

Type TVersionCharSet

Declaration: TVersionCharSet = (vcsASCII, vcsJapan, vcsKorea, vcsTaiwan, vcsUnicode, vcsEasternEuropean, vcsCyrillic, vcsMultilingual, vcsGreek, vcsTurkish, vcsHebrew, vcsArabic, vcsUnknown);

TVersionCharSet is a set of values for the VersionCharSet property of the TVersionInfo class.

Type TVersionLanguage

Declaration: TVersionLanguage = (vlArabic, ..., vlUnknown);

TVersionLanguage is a set of values for the VersionLanguage property of the TVersionInfo class.

Routine OkToWriteModule

Declaration: function OkToWriteModule(ModuleName: string; NewVer: Longint): Boolean;

Return True if it's ok to overwrite ModuleName with NewVer.

OkToWriteModule example:

if OkToWriteModule('C:\WINDOWS\MYLIB.DLL', NewVerNum) then { write file to disk }

Unit Version Information **Comments CompanyName Create FileDescription <u>FileName</u> FileVersion FixedFileInfo <u>GetVerValue</u> InternalName LegalCopyright LegalTrademarks OkToWriteModule OriginalFilename PrivateBuild SpecialBuild Translation TVersionCharSet TVersionLanguage** <u>Valid</u> **VersionCharSet VerLanguage** <u>VerNum</u>

UNIT AppUtils

Unit Overview: GetDefaultIniName GetDefaultIniRegKey GetDefaultSection GetUniqueFileNameInDir **InstantiateForm DefCompanyName RegUseAppTitle AppBroadcast** FindForm **ReadFormPlacement** ReadFormPlacementReg **RestoreFormPlacement** RestoreGridLayout RestoreGridLayoutReg RestoreMDIChildren **RestoreMDIChildrenReg SaveFormPlacement** SaveGridLayout **SaveGridLayoutReg** SaveMDIChildren **SaveMDIChildrenReg** ShowDialog WriteFormPlacement WriteFormPlacementReg

Const DefCompanyName

Declaration: DefCompanyName: string = ";;

This variable uses by GetDefaultIniRegKey function to construct registry key used by <u>TFormPlacement</u> and <u>TFormStorage</u> components.

Const RegUseAppTitle

Declaration: RegUseAppTitle: Boolean = False;;

This variable uses by GetDefaultIniRegKey function to construct registry key used by <u>TFormPlacement</u> and <u>TFormStorage</u> components.

Routine AppBroadcast

Declaration: procedure AppBroadcast(Msg, wParam: Longint; lParam: Longint);

AppBroadcast sends a message Msg to each of the forms of the application.

AppBroadcast example:

```
const CM_MYMESSAGE = WM_USER + 1;
begin
...
AppBroadcast(CM_MYMESSAGE, MyWParam, MyLParam);
...
end;
```

Routine FindForm

Declaration: function FindForm(FormClass: TFormClass): TForm;

FindForm returns the windowed Delphi's form whose class is identified by the specified class name FormClass. If FormClass is not the name of the class of an existing form, FindForm returns nil.

```
FindForm example:
begin
F := FindForm(TMyForm);
if F = nil then
Application.CreateForm(TMyForm, F);
with F do
begin
if WindowState = wsMinimized then WindowState := wsNormal;
Show;
end;
end;
```

Routine FindShowForm

Declaration: function FindShowForm(FormClass: TFormClass; const Caption: string): TForm;

FindShowForm returns the windowed Delphi's form whose class is identified by the specified class name FormClass and (or) window caption passed as Caption. If FormClass is not the name of the class of an existing form, FindShowForm creates new form by calling Application.CreateForm. For found or created form the Show method is called.

FindShowForm example:

```
procedure MainForm.ShowFormItemClick(Sender: TObject);
begin
FindShowForm(TForm1, ");
end:
```

Routine GetDefaultIniName

Declaration: function GetDefaultIniName: string;

Returns the default name for the application initialization file (INI-file). Result string contains the name of the executable application with extension replaced by '.INI' and excluding path information. The result of GetDefaultIniName function can be passed as FileName parameter to the TIniFile constructor.

GetDefaultIniName example:

begin
...
IniFile := TIniFile.Create(GetDefaultIniName);
...
end;

Routine GetDefaultIniRegKey

Declaration: function GetDefaultIniRegKey: string;

Returns the default name for the application registry key. Result string has next format: "HKEY_CURRENT_USER\Software\" + <DefCompanyName\> + <application name> DefCompanyName is global string variable, equal to empty string by default. <application name> is application EXE-name without extension when RegUseAppTitle is False (by default) and is value of Application.Title property when RegUseAppTitle is True. The result of GetDefaultIniRegKey function can be passed as FileName parameter to the TRegIniFile constructor.

GetDefaultIniRegKey example:

begin

```
...
RegUseAppTitle := True;
DefCompanyName := 'My Company';
...
RegIniFile := TRegIniFile.Create(GetDefaultIniRegKey);
...
end;
```

Routine GetDefaultSection

Declaration: function GetDefaultSection(Component: TComponent): string;

The function returns unique string for specified Component. This string can be used, for instance, as section name in the TIniFile (or TRegIniFile) object to store specific component's properties.

Kind of return value can be understand from function's text below:

```
function GetDefaultSection(Component: TComponent): string;
var
F: TCustomForm;
Owner: TComponent;
begin
if Component <> nil then
begin
if Component is TCustomForm then
Result := Component.ClassName
else begin
Result := Component.Name;
```

```
if Component is TControl then
       begin
         F := GetParentForm(TControl(Component));
         if F <> nil then
           Result := F.ClassName + Result
         else begin
           if TControl(Component).Parent <> nil then
              Result := TControl(Component).Parent.Name + Result;
         end;
       end else
       begin
         Owner := Component.Owner;
         if Owner is TForm then
            Result := Format('%s.%s', [Owner.ClassName, Result]);
       end;
    end;
  end else
    Result := ";
end;
GetDefaultSection example:
var
  Ini: TIniFile;
  Sect: string;
begin
  Sect := GetDefaultSection(Bevel1);
  with Bevel1 do
  begin
    Ini.WriteInteger(Sect, 'Shape', Integer(Bevel1.Shape));
    Ini.WriteInteger(Sect, 'Width', Bevel1.Width);
  end;
end;
```

Routine GetUniqueFileNameInDir

Declaration: function GetUniqueFileNameInDir(const Path, FileNameMask: string): string;

The GetUniqueFileNameInDir creates a unique (for specified directory) name for a file. The filename is the concatenation of specified path and strings formed from a specified FileNameMask The FileNameMask parameter **MUST** contain "%d".

GetUniqueFileNameInDir example:

FileName := GetUniqueFileNameInDir('C:\TEMP', 'file%d.tmp');

Routine InstantiateForm

Declaration: function InstantiateForm(FormClass: TFormClass; var Reference): TForm;

When the Reference parameter is nil then InstantiateForm creates a new form of the type specified by the FormClass parameter and assigns it to the variable given by the Reference parameter. The owner of the new form is the Application object. Return value is final value of Reference parameter. When the initial Reference value is not nil then InstantiateForm don't create new form and simply return Reference as function return value.

InstantiateForm example:

with InstantiateForm(TMyForm, MyForm) do ShowModal;

Routine ReadFormPlacement

Declaration: procedure ReadFormPlacement(Form: TForm; IniFile: TIniFile; const Section: string; LoadState, LoadPosition: Boolean);

ReadFormPlacement retrieves state (normal, minimize, maximize) and placement (size and position) of Delphi form Form from an INI file specified by IniFile parameter.

To store form's placement to an INI file use WriteFormPlacement or SaveFormPlacement procedures.

ReadFormPlacement example:

ReadFormPlacement(MyMDIChildForm, IniFile, MyMDIChildForm.ClassName, True, False);

Routine ReadFormPlacementReg

Declaration: procedure ReadFormPlacementReg(Form: TForm; IniFile: TRegIniFile; const Section: string; LoadState, LoadPosition: Boolean);

ReadFormPlacementReg retrieves state (normal, minimize, maximize) and placement (size and position) of Delphi form Form from the Windows 95/NT system registry.

To store form's placement to the system registry use WriteFormPlacementReg procedure.

ReadFormPlacementReg example: ReadFormPlacementReg(MyMDIChildForm, RegIniFile, MyMDIChildForm.ClassName, True, False);

Routine RestoreFormPlacement

Declaration: procedure RestoreFormPlacement(Form: TForm; const IniFileName: string);

NOTE! In 32-bit Delphi versions this procedure has next syntax: procedure RestoreFormPlacement(Form: TForm; const IniFileName: string; UseRegistry: Boolean);

RestoreFormPlacement retrieves state (normal, minimize, maximize) and placement (size and position) of Delphi form Form from an INI file specified by IniFileName parameter (or from the Windows95/NT system registry in 32-bit Delphi versions).

To store form's placement to an INI file or system registry use SaveFormPlacement procedure.

RestoreFormPlacement example:

RestoreFormPlacement(MyForm, GetDefaultIniName);

Routine RestoreGridLayout

Declaration: procedure RestoreGridLayout(Grid: TCustomGrid; IniFile: TIniFile);

Retrieves grid's columns widths and positions (indexes) from an INI file. Can be used for any TCustomGrid descendants such as TStringGrid, TDrawGrid, TRxDrawGrid or other. To save grid's layout to an INI file use SaveGridLayout procedure. You can call this procedure in the OnRestorePlacement event handler of the

TFormPlacement, TFormStorage components.

RestoreGridLayout example: RestoreGridLayout(DrawGrid1, Placement.IniFile);

Routine RestoreGridLayoutReg

Declaration: procedure RestoreGridLayoutReg(Grid: TCustomGrid; IniFile: TRegIniFile);

Retrieves grid's columns widths and positions (indexes) from the Windows95/NT system registry. Can be used for any TCustomGrid descendants such as TStringGrid, TDrawGrid, <u>TRxDrawGrid</u> or other. To save grid's layout to the system registry use SaveGridLayoutReg procedure. You can call this procedure in the OnRestorePlacement event handler of the <u>TFormPlacement</u>, <u>TFormStorage</u> components when its UseRegistry property is True.

RestoreGridLayoutReg example: RestoreGridLayoutReg(DrawGrid1, Placement.RegIniFile);

Routine RestoreMDIChildren

Declaration: procedure RestoreMDIChildren(MainForm: TForm; IniFile: TIniFile);

This procedure restores from an INI-file set and placements of MDI-child forms in MDI application. Use RestoreMDIChildren with SaveMDIChildren to store all child forms in INI-file. You must register all MDI-child form classes by RegisterClass or RegisterClasses procedure to use these classes in SaveMDIChildren and RestoreMDIChildren procedures.

RestoreMDIChildren example:

RestoreMDIChildren(MainForm, IniFile);

initialization RegisterClasses([TChildForm1, TChildForm2]); end.

Routine RestoreMDIChildrenReg

Declaration: procedure RestoreMDIChildrenReg(MainForm: TForm; IniFile: TRegIniFile);

This procedure restores from Windows System Registry set and placements of MDI-child forms in MDI application.

Use RestoreMDIChildrenReg with SaveMDIChildrenReg to store all child forms in Registry.

You must register all MDI-child form classes by RegisterClass or RegisterClasses procedure to use these classes in SaveMDIChildrenReg and RestoreMDIChildrenReg procedures.

RestoreMDIChildrenReg example:

RestoreMDIChildrenReg(MainForm, RegIniFile);

... initialization RegisterClasses([TChildForm1, TChildForm2]); end.

Routine SaveFormPlacement

Declaration: procedure SaveFormPlacement(Form: TForm; const IniFileName: string);

NOTE! In 32-bit Delphi versions this procedure has next syntax:

procedure SaveFormPlacement(Form: TForm; const IniFileName: string; UseRegistry: Boolean);

SaveFormPlacement writes state (normal, minimize, maximize) and placement (size and position) of Delphi form Form to an INI file specified by IniFileName parameter (or to the Windows95/NT system registry in 32-bit Delphi versions).

To retrieve form's placement from an INI file or system registry use RestoreFormPlacement procedure.

SaveFormPlacement example:

SaveFormPlacement(MyForm, GetDefaultIniName);

Routine SaveGridLayout

Declaration: procedure SaveGridLayout(Grid: TCustomGrid; IniFile: TIniFile);

Write grid's columns widths and positions (indexes) to an INI file. Can be used for any TCustomGrid descendants such as TStringGrid, TDrawGrid, TRxDrawGrid or other. To retrieve grid's layout from an INI file use RestoreGridLayout procedure. You can call this procedure in the OnSavePlacement event handler of the TFormPlacement, TFormStorage components.

SaveGridLayout example:

SaveGridLayout(DrawGrid1, Placement.IniFile);

Routine SaveGridLayoutReg

Declaration: procedure SaveGridLayoutReg(Grid: TCustomGrid; IniFile: TRegIniFile);

Writes grid's columns widths and positions (indexes) to the Windows95/NT system registry. Can be used for any TCustomGrid descendants such as TStringGrid, TDrawGrid, TRxDrawGrid or other. To retrieve grid's layout from the system registry use RestoreGridLayoutReg procedure. You can call this procedure in the OnSavePlacement event handler of the TFormPlacement, TFormStorage components when its UseRegistry property is True.

SaveGridLayoutReg example:

SaveGridLayoutReg(DrawGrid1, Placement.RegIniFile);

Routine SaveMDIChildren

Declaration: procedure SaveMDIChildren(MainForm: TForm; IniFile: TIniFile);

This procedure stores in INI-file set and placements of opened MDI-child forms in MDI application. Use SaveMDIChildren with RestoreMDIChildren to open all child forms stored by SaveMDIChildren. You must register all MDI-child form classes by RegisterClass or RegisterClasses procedure to use these classes in SaveMDIChildren and RestoreMDIChildren procedures.

SaveMDIChildren example:

SaveMDIChildren(MainForm, IniFile);

Routine SaveMDIChildrenReg

Declaration: procedure SaveMDIChildrenReg(MainForm: TForm; IniFile: TRegIniFile);

This procedure stores in Windows System Registry set and placements of opened MDI-child forms in MDI application.

Use SaveMDIChildrenReg with RestoreMDIChildrenReg to open all child forms stored by SaveMDIChildrenReg.

You must register all MDI-child form classes by RegisterClass or RegisterClasses procedure to use these classes in SaveMDIChildrenReg and RestoreMDIChildrenReg procedures.

SaveMDIChildrenReg example:

SaveMDIChildrenReg(MainForm, RegIniFile);

Routine ShowDialog

Declaration: function ShowDialog(FormClass: TFormClass): Boolean;

ShowDialog creates Delphi form of the type specified by the FormClass parameter and shows new form as a modal form by using ShowModal method. When the modal form is closed, ShowDialog destroys the form by using Free method.

ShowDialog returns True if the ModalResult property of the form is mrOk or mrYes, another ShowDialog returns False.

ShowDialog example:

var Ok: Boolean; begin Ok := ShowDialog(TAboutDlg); end;

Routine WriteFormPlacement

Declaration: procedure WriteFormPlacement(Form: TForm; IniFile: TIniFile; const Section: string);

WriteFormPlacement writes state (normal, minimize, maximize) and placement (size and position) of Delphi form Form to an INI file specified by IniFileName parameter. To retrieve form's placement from an INI file use ReadFormPlacement procedure.

WriteFormPlacement example:

WriteFormPlacement(MyForm, IniFile, MyForm.ClassName);

Routine WriteFormPlacementReg

Declaration: procedure WriteFormPlacementReg(Form: TForm; IniFile: TRegIniFile; const Section: string);

WriteFormPlacementReg writes state (normal, minimize, maximize) and placement (size and position) of Delphi form Form to the Windows95/NT system registry key specified by IniFile parameter. To retrieve form's placement from the system registry use ReadFormPlacementReg procedure.

WriteFormPlacementReg example:

WriteFormPlacementReg(MyForm, RegIniFile, MyForm.ClassName);

Control Name	Unit	Class
BDE Utilities	BdeUtils	TDBLocate

Description:

Use TDBLocate to search a BDE-dataset specified by DataSet property for a specific record and position the cursor on it.

After setting DataSet property you can call Locate method which has parameter KeyField specified field name on which to search.

When you search on non-BDE data source, use TLocateObject class instead.

TDBLocate uses the fastest possible method to locate matching records. If DataSet specifies the TTable object and the search field is a key field and the IndexSwitch property is True, TDBLocate uses the index. Otherwise TDBLocate creates a filter for the search if applicable or use full search throw dataset otherwise.

NOTE. In Delphi 2.0 or higher you can use standard TDataSet.Locate method instead.

Property DataSet

Declaration: DataSet: TDataSet;

Specifies the dataset (table or query) for which Locate method will search for specified value. Set this property and IndexSwitch property to appropriate values before call Locate method.

Routine AsyncQrySupported

Declaration: function AsyncQrySupported(Database: TDatabase): Boolean;

Returns True if database driver specified by Database parameter supports asynchronous query execution. In 32-bit Delphi versions you can use OnServerYield event when True returned by AsyncQrySupported (for example, to break query execution in multi-threaded application). Currently, only Sybase driver supports asynchronous query execution.

AsyncQrySupported example:

When you execute your queries in separate thread, you can set OnServerYield event handler, for example, to end query processing.

You can check database for supporting yields control during query processing to enable or disable appropriate menu item:

CancelQueryItem.Visible := AsyncQrySupported(Database);

Routine CheckOpen

Declaration: function CheckOpen(Status: DBIResult): Boolean;

Checks the result of a call to the Borland Database Engine (BDE). Call CheckOpen to determine if a call to the BDE returns an error when an attempt is made to access a table. Status is the return result of a previous call to the BDE. CheckOpen returns True if there is access is successful. If Status indicates insufficient table rights when accessing a Paradox table, CheckOpen calls the database session's GetPassword method to prompt the user for a password. If the dialog is successful, CheckOpen returns True. Otherwise CheckOpen returns False, indicating that dataset access failed.

```
> CheckOpen example:
```

```
function TFieldList.CreateHandle: HDBICur;
var
STableName: PChar;
begin
STableName := StrAlloc(Length(FTableName) + 1);
try
while not CheckOpen(DbiOpenFieldList(DBHandle, AnsiToNative(DBLocale,
FTableName, STableName, Length(FTableName)), nil, False, Result)) do
{Retry};
finally
StrDispose(STableName);
end;
end;
```

Routine ConvertStringToLogicType

Declaration: procedure ConvertStringToLogicType(Locale: TLocale; FldLogicType: Integer; FldSize: Word; const FldName, Value: String; Buffer: Pointer);

ConvertStringToLogicType translates string specified by Value parameter into BDE logical type specified by FldLogicType. You can use translated value when you call BDE API functions directly.

You must allocate memory for Buffer to retrieve translated value. FldSize parameter specifies the maximum number of bytes to copy to the buffer.

ConvertStringToLogicType example:

var DataSet: TDataSet; Fld: TField; Buffer: Pointer; FldName: string; begin

```
...

Fld := DataSet.FieldByName(FldName);

GetMem(Buffer, Fld.DataSize);

try

ConvertStringToLogicType(DataSet.Locale, FieldLogicMap(Fld.DataType),

Fld.DataSize, FldName, Value, Buffer);

{ use the value in Buffer }

finally

FreeMem(Buffer, Fld.DataSize);

end;

end;
```

Routine CurrentRecordDeleted

Declaration: function CurrentRecordDeleted(DataSet: TBDEDataSet): Boolean;

Determines if the current record of DataSet is deleted. Applicable only when soft delete is supported (dBASE and FoxPro only) and soft deletes is set to True by calling DatSetShowDeleted procedure.

CurrentRecordDeleted example:

if CurrentRecordDeleted(Table1) then
begin
{ undelete record }
end;

Routine DataSetFindValue

Declaration: function DataSetFindValue(DataSet: TDataSet; const Value, FieldName: string): Boolean;

Searches the dataset for a specified record and makes that record the current record.

Call DataSetFindValue to search a dataset for a specific record and position the cursor on it. FieldName is a string containing a field name on which to search. Value is a value to match in the key field.

DataSetFindValue returns True if it finds a matching record, and makes that record the current one. Otherwise DataSetFindValue returns False.

DataSetFindValue example:

var DataSet: TDataSet; FieldName: string; Found: Boolean; begin

Found := DataSetFindValue(DataSet, DataSet.FieldByName(FieldName).AsString, FieldName); if not Found then

MessageDlg('Record not found', mtError, [mbOk], 0);

end;

•••

Routine DataSetPositionStr

Declaration: function DataSetPositionStr(DataSet: TDataSet): string;

Retrieves string with current record number and full records count in DataSet in following format: <record_no> : <record_count>.

This function is supported for Paradox and DBase drivers only.

DataSetPositionStr example:

procedure TMDIChild.DataSource1DataChange(Sender: TObject; Field: TField); begin

Panel1.Caption := DataSetPositionStr(DataSource1.DataSet);

Routine DataSetRecNo

Declaration: function DataSetRecNo(DataSet: TDataSet): Longint;

This function retrieves the sequence (for Paradox and In-Memory tables) or physical (for DBase tables) number of the current record in the dataset DataSet. Applications might use this function with RecordCount property to iterate through all the records in a dataset, though typically record iteration is handled with calls to First, Last, MoveBy, Next, and Prior.

DataSetRecNo example:

RecNo := DataSetRecNo(Table1);

Routine DataSetShowDeleted

Declaration: procedure DataSetShowDeleted(DataSet: TBDEDataSet; Show: Boolean);

Toggles mode to show or hide deleted records in DataSet. Applicable only when soft delete is supported (dBASE and FoxPro only).

DataSetShowDeleted example:

DataSetShowDeleted(Table1, True);

Routine DeleteRange

Declaration: procedure DeleteRange(Table: TTable; IndexFields: array of const; FieldValues: array of const);

Deletes from Table all records specified by the key values FieldValues. Table must have an index on fields specified by IndexFields parameters.

DeleteRange example:

DeleteRange(Table1, ['ORDER_ID', 'CUSTOMER_NO'], [1, 255]);

Routine ExecuteQuery

Declaration: procedure ExecuteQuery(const DbName, QueryText: string);

Executes the SQL statement QueryText in the database specified by DbName parameter.

ExecuteQuery example:

ExecuteQuery('DBDEMOS', 'DELETE FROM CUSTOMER WHERE CustNo = 2135');

Routine ExportDataSet

Declaration: procedure ExportDataSet(Source: TDataSet; DestTable: TTable; TableType: TTableType; const

end;

AsciiCharSet: string; AsciiDelimited: Boolean; MaxRecordCount: Longint);

This procedure exports data from data source specified by Source parameter to the table specified by DestTable parameter. Function create the destination table based on the structure of the source data set. If the destination already exists, the function will delete it, and replace it with the new copy of the source.

For export into ASCII-format you can specify destination character set by AsciiCharSet parameter and fixed or delimited (varying) format by AsciiDelimited parameter. RecordCount parameter specifies the maximum number of records that are exported to the destination table when function is called. If zero, all records are added, beginning with the first record in Source.

ExportDataSet example:

var SourceDataSet: TDataSet; DestName, CharSet: string; begin {...} if (SourceDataSet <> nil) then begin if SourceDataSet.Active then SourceDataSet.CheckBrowseMode; if (SourceDataSet is TTable) then DestName := ExtractFileName(TTable(SourceDataSet).TableName) else begin if not SourceDataSet.Active then DBError(SDataSetClosed); DestName := 'Query'; end: DestName := ChangeFileExt(DestName, '.TXT'); end; DestTable.TableName := DestName; ExportDataSet(SourceDataSet, DestTable, ttASCII, 'db866ru0', True, 0); MessageDlg(Format('Table %s successfully created.', [DestTable.TableName]), mtInformation, [mbOk], 0); end;

Routine FetchAllRecords

Declaration: procedure FetchAllRecords(DataSet: TDataSet);

The FetchAllRecords procedure, when used with a query against a server database, forces the server to release all intermediate locks and reads the entire result set.

In Delphi 2.x or higher, use TDataset.FetchAll method instead.

FetchAllRecords example:

```
Query1.Open;
try
FetchAllRecords(Query1);
{ working with query result set }
finally
Query1.Close;
end;
```

Routine FieldLogicMap

Declaration: function FieldLogicMap(FldType: TFieldType): Integer;

Returns BDE logical type identifier for the specified Delphi database field type.

FieldLogicMap example:

var I: Integer; Fld: TField; begin ... I := FieldLogicMap(Fld.DataType); end;

Routine GetAliasPath

Declaration: function GetAliasPath(const AliasName: string): string;

Retrieves physical path for BDE alias specified by AliasName parameter. If specified database is SQL-bases database, the function returns server name as result.

> GetAliasPath example:

S := Format('Database path: %s', GetAliasPath(Database.AliasName));

Routine GetBDEDirectory

Declaration: function GetBDEDirectory: string;

Retrieves the path of the Borland Database Engine (BDE) directory. The BDE directory contains all BDE DLLs.

> GetBDEDirectory example: ShowMessage(Format('BDI installed at %s', [GetBDEDirectory]));

Routine InitRSRun

Declaration: procedure InitRSRun(Database: TDatabase; const ConName: string; ConType: Integer; const ConServer: string);

Initializes Report Smith runtime by updating RPTSMITH.CON file.

ConName parameter specifies connection name. You can use empty string as this parameter to use application EXE-name as connection name. ConType is a connection type, defined in REPORT.PAS unit (i.e. ctIDAPIDBase). ConServer specifies server name, for example, 'PARADOX' or 'IBLOCAL'.

InitRSRun example:
procedure TMainForm.FormCreate(Sender: TObject); begin InitRSRUN(Database, ", ctIDAPIParadox, 'PARADOX'); end;

Routine IsBookmarkStable

Declaration: function IsBookmarkStable(DataSet: TDataSet): Boolean;

Return True, if specified DataSet supports stable bookmarks. Stable bookmarks are those that remain unchanged after another user has modified the table. For example, return value is True for Paradox tables having a primary key, but False for Paradox heap tables.

IsBookmarkStable example:

```
BM := Table1.GetBookmark;

try

{...}

if IsBookmarkStable(Table1) then

SetToBookmark(Table1, BM); { no exceptions raised }

finally

Table1.FreeBookmark(BM);

end;
```

Routine PackTable

Declaration: procedure PackTable(Table: TTable);

Packs the specified table, i.e deletes unused records from a Paradox or dBase table. This function does not work with SQL databases.

PackTable example:

PackTable(Table1);

Routine RestoreIndex

Declaration: procedure RestoreIndex(Table: TTable);

Restores TTable.IndexFieldNames property after calling SetIndex procedure. It's important to use SetIndex and RestoreIndex in conjunction. > RestoreIndex example:

SetIndex(Table1, 'NAME');

```
try
...
finally
RestoreIndex(Table1);
end;
```

Routine SetIndex

Declaration: procedure SetIndex(Table: TTable; const IndexFieldNames: string);

Use SetIndex specify an alternative index for a table by setting IndexFieldNames property. Use this procedure only with RestoreIndex:

```
SetIndex(Table1, 'NAME');
try
...
finally
RestoreIndex(Table1);
end;
```

SetIndex example:

```
SetIndex(Table1, 'NAME');
try
...
finally
RestoreIndex(Table1);
end;
```

Routine SetToBookmark

Declaration: function SetToBookmark(ADataSet: TDataSet; ABookmark: TBookmark): Boolean;

Returns dataset specified by the ADataSet parameter to a bookmark previously created by GetBookmark method. SetToBookmark returns True if the call was successful, otherwise it returns False. Note: If the bookmark is unstable, the cursor may be in an unexpected position.

SetToBookmark example:

BM := Table1.GetBookmark; try {...} SetToBookmark(Table1, BM); { no exceptions raised } finally Table1.FreeBookmark(BM); end;

Routine TransActive

Declaration: function TransActive(Database: TDatabase): Boolean;

Indicates whether a database transaction is in progress or not. Use this function to determine if a database transaction is currently in progress. Return value is True if a transaction is in progress, False otherwise.

TransActive example:

if TransActive(Database) then Database.Commit;

Unit <u>BdeUtils</u> Property <u>DataSet</u>

Routine **AsyncQrySupported CheckOpen ConvertStringToLogicType CurrentRecordDeleted DataSetFindValue DataSetPositionStr DataSetRecNo DataSetShowDeleted DeleteRange ExecuteQuery ExportDataSet FetchAllRecords FieldLogicMap GetAliasPath GetBDEDirectory InitRSRun IsBookmarkStable PackTable RestoreIndex** SetIndex SetToBookmark **TransActive**

Control Name	Unit	Class
Locate Object	DBUtils	TLocateObject

Description:

Use TLocateObject to search a dataset specified by DataSet property for a specific record and position the cursor on it.

After setting <u>DataSet</u> property you can call <u>Locate</u> method which has parameter KeyField specified field name on which to search.

When you search on BDE data source, use TDBLocate class (inherits from TLocateObject) instead.

NOTE. In Delphi 2.0 or higher you can use standard TDataSet.Locate method instead.

Property DataSet

Declaration: DataSet: TDataSet;

Specifies the dataset (table or query) for which Locate method will search for specified value. Set this property and IndexSwitch property to appropriate values before call Locate method.

Property IndexSwitch

Declaration: IndexSwitch: Boolean;

If the dataset specified by DataSet property is TTable object and search field is indexed, TDBLocate can use the index. This property determines whether or not table will switch to the index corresponding to KeyField parameter of Locate method. When this property is False, index is used only when it's active index.

This property used by TDBLocate class only. Setting this property in TLocateObject class has no effect.

Method Locate

Declaration: function Locate(const KeyField, KeyValue: string; Exact, CaseSensitive: Boolean): Boolean;

Searches the dataset specified by DataSet property for a specific record and position the cursor on it. KeyField parameter is a string specified field name on which to search. KeyValue is a string containing the value to match in the key field.

If CaseSensitive parameter is False, then Locate ignores case when matching string fields. If Exact parameter is False, then Locate finds the first record that fulfills at least some initial part of the KeyValue criteria for record matching.

<u>Locate</u> returns True if it finds a matching record, and makes that record the current one. Otherwise Locate returns False.

Declaration: ServerDateFmt: string[50] = """"mm"/"dd"/"yyyy""";;

ServerDateFmt variable specifies date format used in SQL queries by function FormatSQLDateRange, FormatSQLCondition and FormatAnsiSQLCondition. You can assign another value to this variable according to date format used by your SQL Server.

DBUTILS.PAS unit also contains additional constants that can be assigned to the ServerDateFmt variable:

- sdfStandard16, sdfStandard32 date format for STANDARD driver (local SQL);
- sdfOracle date format for using with Oracle SQL-server;
- sdfInterbase date format for using with Interbase SQL-server.

Routine AssignRecord

Declaration: procedure AssignRecord(Source, Dest: TDataSet; ByName: Boolean);

AssignRecord copies values of fields from current record of source dataset specified by Source parameter to the current records of dataset Dest. Destination dataset must be in edit or insert mode.

When ByName parameter is True, then field values will be set from Source to Dest based on their names in the both datasets. If ByName is False, field values will be set based on the order in which fields are defined in the Source dataset.

AssignRecord example:

MemoryTable1.Append; AssignRecord(SourceDataSet, MemoryTable1, True); MemoryTable1.Post;

Routine CheckRequiredField

Declaration: procedure CheckRequiredField(Field: TField);

Checks if a field Field has a nonblank value. Calling this function with a null value of Field will cause an exception to be raised.

CheckRequiredField example:

CheckRequiredField(InventoryTableACCOUNT_NO);

Routine ConfirmDataSetCancel

Declaration: procedure ConfirmDataSetCancel(DataSet: TDataSet);

The ConfirmDatasetCancel procedure verifies that the dataset's state, and if the dataset's State property is dsEdit or dsInsert displays a message box which asks user for confirmation that the pending changes will be saved to the database or will be cancelled. If user selected "Yse" (save) in a message box, the dataset's Post method is called to post any pending changes, if user selected "No", the dataset's Cancel method is called, otherwise (if the "Cancel" button was selected by user) the standard Abort procedure is called to generate EAbort exception.

ConfirmDataSetCancel example:

procedure TForm1.FormCloseQuery(Sender: TObject; var CanClose: Boolean); begin ConfirmDataSetCancel(Table1); end;

Routine ConfirmDelete

Declaration: function ConfirmDelete: Boolean;

The ConfirmDelete procedure displays a message box which asks user for confirmation that the record from dataset should really be deleted. The function returns True if user selected "Yes" button in a message box. Otherwise the function returns False.

ConfirmDelete example:

if ConfirmDelete then Table1.Delete;

Routine DataSetSortedSearch

Declaration: function DataSetSortedSearch(DataSet: TDataSet; const Value: string; const FieldName: string; Unique, IgnoreCase: Boolean): Boolean;

The DataSeetSortedSearch procedure searches a *dataset* for *value* in the field *FieldName*.

DataSetSortedSearch example:

```
if not DataSetSortedSearch(DataSet, 'Smit', 'Client_Name', False, True) then
MessageDlg('Record not found', mtError, [mbOk], 0);
```

•••

Routine FormatAnsiSQLCondition

Declaration: function FormatAnsiSQLCondition(const FieldName, Operator, Value: string; FieldType: TFieldType; Exact: Boolean): string;

FormatAnsiSQLCondition formats an ANSI SQL condition to the BDE equivalent.

FormatAnsiSQLCondition example:

rxQuery1.MacroByName('CONTRACT_NO').AsString :=
FormatSQLCondition('CONTRACTS."CONTRACT_NO''', ", 'N%4596*', ftString, True);

Routine FormatSQLCondition

Declaration: function FormatSQLCondition(const FieldName, Operator, Value: string; FieldType: TFieldType; Exact: Boolean): string;

FormatSQLCondition formats an SQL condition to the BDE equivalent.

FormatSQLCondition example:

rxQuery1.MacroByName('CONTRACT_NO').AsString :=
FormatSQLCondition('CONTRACTS.''CONTRACT_NO''', ", 'N-4596?', ftString, True);

Routine FormatSQLDateRange

Declaration: function FormatSQLDateRange(Date1, Date2: TDateTime; const FieldName: string): string;

FormatSQLDateRange checks a fieldname for a date period within Date1 and Date2.

FormatSQLDateRange example:

FormatSQLDateRange(0, Date1, 'CONTRACTS."ACTIVE_FROM"') Date1 = 01.01.95 Result = 'CONTRACTS."ACTIVE_FROM'' < "01/01/1995"'

Routine RefreshQuery

Declaration: procedure RefreshQuery(Query: TDataSet);

The RefreshQuery procedure for a dataset Query flushes local buffers and refetches data for an open dataset. You can use this procedure to update the display in data-aware controls if you think that the underlying data has changed because other applications have simultaneous access to the data used in your application.

RefreshQuery example:

RefreshQuery(Query1);

Routine RestoreFields

Declaration: procedure RestoreFields(DataSet: TDataSet; IniFile: TIniFile; RestoreVisible: Boolean);

RestoreFields procedure restores widths (in pixels) and indexes of dataset's fields from the INI-file specified in IniFile parameter, previously stored by SaveFields procedure. Can be used to save and restore field's parameters changed by user, for example, in a DBGrid component.

RestoreFields example:

```
procedure RestoreLayout;
var
IniFile: TIniFile;
begin
IniFile := TIniFile.Create(FormStorage.IniFileName);
try
RestoreFields(DataSet, IniFile);
finally
IniFile.Free;
end;
end;
```

Routine RestoreFieldsReg

Declaration: procedure RestoreFieldsReg(DataSet: TDataSet; IniFile: TRegIniFile; RestoreVisible: Boolean);

RestoreFieldsReg procedure restores widths (in pixels) and indexes of dataset's fields from the Windows

System Registry key specified in IniFile parameter, previously stored by SaveFieldsReg procedure. Can be used to save and restore field's parameters changed by user, for example, in a DBGrid component.

RestoreFieldsReg example:

```
uses Registry, Placemnt, DBUtils;
...
procedure RestoreDSLayout;
var
RegIniFile: TRegIniFile;
begin
IniFile := TRegIniFile.Create(FormStorage.IniFileName);
try
RestoreFieldsReg(DataSet, RegIniFile);
finally
RegIniFile.Free;
end;
end;
```

Routine SaveFields

Declaration: procedure SaveFields(DataSet: TDataSet; IniFile: TIniFile);

SaveFields procedure saves widths (in pixels) and indexes of dataset's fields to the INI-file specified in IniFile parameter.

Can be used to store field's parameters changed by user, for example, in a DBGrid component. To restore widths and indexes of fields use RestoreFields procedure.

SaveFields example:

```
procedure TGridForm.FormStorageSavePlacement(Sender: TObject);
var
DataSet: TDataSet; I: Integer;
begin
for I := 0 to TabSet.Tabs.Count - 1 do
begin
DataSet := DataSetByIndex(TabSet.TabIndex);
if (DataSet <> nil) and (DataSet.Active) then
SaveFields(DataSet, FormStorage.IniFile);
end;
end;
```

Routine SaveFieldsReg

Declaration: procedure SaveFieldsReg(DataSet: TDataSet; IniFile: TRegIniFile);

SaveFieldsReg procedure saves widths (in pixels) and indexes of dataset's fields to the Windows

System Registry in the key specified in IniFile parameter. Can be used to store field's parameters changed by user, for example, in a DBGrid component. To restore widths and indexes of fields use RestoreFieldsReg procedure.

SaveFieldsReg example:

```
procedure TGridForm.FormStorageSavePlacement(Sender: TObject);
var
DataSet: TDataSet; I: Integer;
begin
for I := 0 to TabSet.Tabs.Count - 1 do
begin
DataSet := DataSetByIndex(TabSet.TabIndex);
if (DataSet <> nil) and (DataSet.Active) then
if FormStorage.UseRegistry then
SaveFieldsReg(DataSet, FormStorage.RegIniFile)
else
SaveFields(DataSet, FormStorage.IniFile);
end;
end;
```

Unit **DBUtils**

Property DataSet IndexSwitch

Method Locate

Const <u>ServerDateFmt</u>

Routine AssignRecord CheckRequiredField ConfirmDataSetCancel ConfirmDelete DataSetSortedSearch FormatAnsiSQLCondition FormatSQLCondition FormatSQLDateRange RefreshQuery RestoreFields RestoreFieldsReg SaveFieldsReg

UNIT BoxProcs

Routine <u>BoxDragOver</u> Routine <u>BoxMoveAllItems</u> Routine <u>BoxMoveFocusedItem</u> Routine <u>BoxMoveFocusedItem</u>

This is the low level code to manipulate the **<u>dual dialog box</u>**.

Routine BoxDragOver

Declaration: procedure BoxDragOver(List: TCustomListBox; Source: TObject; X, Y: Integer; State: TDragState; var Accept: Boolean; Sorted: Boolean);

This is Intended to be used in the *OnDragOver* event handlers.

BoxDragOver example:

procedure TMyForm.MyListDragOver(Sender, Source: TObject; X, Y: Integer; State: TDragState; var Accept: Boolean); begin BoxDragOver(MyList, Source, X, Y, State, Accept, MyList.Sorted); if State = dsDragLeave then (Source as TListBox).DragCursor := crDrag; if (State = dsDragEnter) and ((Source as TListBox).SelCount > 1) then (Source as TListBox).DragCursor := crMultiDrag; end:

Routine BoxMoveAllItems

Declaration: procedure BoxMoveAllItems(SrcList, DstList: TCustomListBox);

Copies all items from SrcList to the DstList and then clears the SrcList.

BoxMoveAllItems example:

BoxMoveAllItems(DstList, SrcList);

Routine BoxMoveFocusedItem

Declaration: procedure BoxMoveFocusedItem(List: TCustomListBox; DstIndex: Integer);

Indended to be used in the OnDragDrop event handlers.

BoxMoveFocusedItem example:

procedure TMyForm.SrcListDragDrop(Sender, Source: TObject; X, Y: Integer); begin if Source = DstList then ExclBtnClick(SrcList) else

```
if Source = SrcList then
begin
BoxMoveFocusedItem(SrcList, SrcList.ItemAtPos(Point(X, Y), True));
end;
end;
```

Routine BoxMoveSelectedItems

Declaration: procedure BoxMoveSelectedItems(SrcList, DstList: TCustomListBox);

Moves all selected items from the SrcList to the DstList.

BoxMoveSelectedItems example: procedure TMyForm.IncBtnClick(Sender: TObject); begin BoxMoveSelectedItems(SrcList, DstList); end;

UNIT

DateUtil

Const FourDigitYear

Туре

<u>TDateOrder</u> <u>TDayOfWeekName</u> <u>TDaysOfWeek</u>

Routine

CutTime DateDiff **DaysBetween** DaysInPeriod DaysPerMonth DefDateFormat DefDateMask IncDate IncDay IncHour **IncMinute** IncMonth IncMSec **IncTime** IncYear **IsLeapYear** LastDayOfPrevMonth MonthsBetween StrToDateDef **StrToDateFmt** <u>StrToDateFmtDef</u> ValidDate

Const FourDigitYear

Declaration: const FourDigitYear: Boolean = True;;

To control 2- or 4-digit year in TDateEdit, TDBDateEdit components you can use a typed constant FourDigitYear. The code in initialization section of DateUtil.pas is setting this constant to True or False depending on ShortDateFormat variable. You can set FourDigitYear to True in your project code or in initialization section of your unit (before any of TDateEdit components created) to ensure that your users will be able to enter the year that they need.

When FourDigitYear is False the date entered is always treated as date in the current century so 01 entered now is 1901 but 01 entered in 2000 is 2001.

Type TDateOrder

Declaration: TDateOrder = (doMDY, doDMY, doYMD);

An enumerated type used to represent a possible day's, month's and year's order in the date format.

Type TDayOfWeekName

Declaration: TDayOfWeekName = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);

An enumerated type used when representing a day of the week.

Type TDaysOfWeek

Declaration: TDaysOfWeek = set of TDayNameOfWeek;

Set of day of the week.

Routine CutTime

Declaration: function CutTime(ADate: TDateTime): TDateTime;

Set time in the ADate parameter to 00:00:00:00.

CutTime example: DateWithoutTime := CutTime(ADate);

Routine DateDiff

Declaration: procedure DateDiff(Date1, Date2: TDateTime; var Days, Months, Years: Word);

Return the difference in days, months, and years between two valid dates Date1 and Date2.

DateDiff example:

```
var
D1, D2: TDateTime;
D, M, Y: Word;
begin
ShortDateFormat := 'dd.mm.yyyy';
D1 := StrToDate('24.03.1994');
D2 := StrToDate('12.08.1995');
DateDiff(D1, D2, D, M, Y);
{ here D=19; M=4; Y=1 }
end;
```

Routine DaysBetween

Declaration: function DaysBetween(Date1, Date2: TDateTime): Longint;

Count days between Date1 and Date2 + 1. If Date1 = Date2 then result = 1, if Date2 < Date1 result = 0.

DaysBetween example:

Days := DaysBetween(Date1, Date2);

Routine DaysInPeriod

Declaration: function DaysInPeriod(Date1, Date2: TDateTime): Longint;

Count days between Date1 and Date2 + 1, so if Date1 = Date2 result = 1.

> DaysInPeriod example: Days := DaysInPeriod(Date1, Date2);

Routine DaysPerMonth

Declaration: function DaysPerMonth(AYear, AMonth: Integer): Integer;

DaysPerMonth returns the number of days in the specified month AMonth (1..12) and year AYear.

DaysPerMonth example:

The following example returns 28, since 1995 is not a leap year.

var D: Word; begin D := DaysPerMonth(1995, 2); end;

Routine DefDateFormat

Declaration: function DefDateFormat: string;

DefDateFormat returns the default date format string according to the current Windows settings (ShortDateFormat) and to the FourDigitYear variable's value. The order for month, day, and year is determined by the ShortDateFormat global variable.

For example, if the ShortDateFormat = 'D/mm/yy', then the DefDateFormat returns: 'DD/MM/YYYY'.

> DefDateFormat example:

Field1.AsDateTime := StrToDateFmt(DefDateFormat, Edit1.Text);

Routine DefDateMask

Declaration: function DefDateMask(BlanksChar: Char): string;

DefDateMask returns the default edit mask for the date (for using as EditMask property value in the TMaskEdit component) according to the current Windows settings (ShortDateFormat) and to the FourDigitYear variable's value. The order for month, day, and year is determined by the ShortDateFormat global variable.

For example, if the ShortDateFormat = 'D/mm/yy', then the DefDateMask returns: '!99/99/9999;1; '.

> DefDateMask example:

MaskEdit1.EditMask := DefDateMask('_');

Routine FirstDayOfNextMonth

Declaration: function FirstDayOfNextMonth: TDateTime;

Returns the first day of the next month relative to the current system date.

> FirstDayOfNextMonth example:

First := FirstDayOfNextMonth;

Routine FirstDayOfPrevMonth

Declaration: function FirstDayOfPrevMonth: TDateTime;

Returns the first day of the previous month relative to the current system date.

> FirstDayOfPrevMonth example:

First := FirstDayOfPrevMonth;

Routine GetDateOrder

Declaration: function GetDateOrder(const DateFormat: string): TDateOrder;

Returns the day's, month's and year's order in the date format, indicated by the DateFormat parameter.

> GetDateOrder example:

case GetDateOrder(ShortDateFormat) of doMDY: Result := 'MM/DD/YYYY'; doDMY: Result := 'DD/MM/YYYY'; doYMD: Result := 'YYYY/MM/DD'; end;

Routine IncDate

Declaration: function IncDate(ADate: TDateTime; Days, Months, Years: Integer): TDateTime;

IncDate adjusts a date by the specified number of days, months, and years. IncDate adds (or subtracts) the specified number of Days, Months, and Years to (or from) a date ADate.

> IncDate example:

var D1, D2: TDateTime; S: string; begin ShortDateFormat := 'dd.mm.yy'; D1 := StrToDate('24.03.1994'); D2 := IncDate(D1, 38, 5, 2); S := DateToStr(D2); { here S = '01.10.96' } end;

Routine IncDay

Declaration: function IncDay(ADate: TDateTime; Delta: Integer): TDateTime;

IncDay changes the date ADate by Delta number of days. Delta can be either a positive or negative value.

> IncDay example:

var D1, D2: TDateTime; S: string; begin ShortDateFormat := 'dd.mm.yy'; D1 := StrToDate('24.03.1994'); D2 := IncDay(D1, 145); S := DateToStr(D2); end;

Routine IncHour

Declaration: function IncHour(ATime: TDateTime; Delta: Integer): TDateTime;

IncHour changes the time of day ATime by Delta number of hours. Delta can be either a positive or negative value.

> IncHour example:

var T1, T2: TDateTime; S: string; begin ShortTimeFormat := 'HH:MM:SS'; T1 := StrToTime('00:00:00'); T2 := IncHour(T1, -1); S := TimeToStr(T2); end;

Routine IncMinute

Declaration: function IncMinute(ATime: TDateTime; Delta: Integer): TDateTime;

IncMinute changes the time of day ATime by Delta number of minutes. Delta can be either a positive or negative value.

> IncMinute example:

var T1, T2: TDateTime; S: string; begin ShortTimeFormat := 'HH:MM:SS'; T1 := StrToTime('00:00:00'); T2 := IncMinute(T1, 5); S := TimeToStr(T2); end;

Routine IncMonth

Declaration: function IncMonth(ADate: TDateTime; Delta: Integer): TDateTime;

IncMonth changes the date ADate by Delta number of months. Delta can be either a positive or negative value.

> IncMonth example:

var D1, D2: TDateTime; S: string; begin ShortDateFormat := 'dd.mm.yy'; D1 := StrToDate('30.12.1991'); D2 := IncMonth(D1, 2); S := DateToStr(D2); end;

Routine IncMSec

Declaration: function IncMSec(ATime: TDateTime; Delta: Integer): TDateTime;

IncMSec changes the time of day ATime by Delta number of milliseconds. Delta can be either a positive or negative value.

> IncMSec example:

var T: TDateTime; Hour, Min, Sec, MSec; begin T := IncMSec(0, -100); DecodeTime(T, Hour, Min, Sec, MSec); { here Hour = 23, Min = 59, Sec = 59, MSec = 900 } end;

Routine IncSecond

Declaration: function IncSecond(ATime: TDateTime; Delta: Integer): TDateTime;

IncSecond changes the time of day ATime by Delta number of seconds. Delta can be either a positive or negative value.

> IncSecond example:

var T1, T2: TDateTime; S: string; begin ShortTimeFormat := 'HH:MM:SS'; T1 := StrToTime('00:00:00'); T2 := IncSecond(T1, -10); S := TimeToStr(T2); end;

Routine IncTime

Declaration: function IncTime(ATime: TDateTime; Hours, Minutes, Seconds, MSecs: Integer): TDateTime;

IncTime adds the specified hours, minutes, seconds and milliseconds to the specified time of day ATime. The result of IncTime is adjusted to account for the rollover at midnight.

> IncTime example:

```
var T1, T2: TDateTime; S: string;
begin
ShortTimeFormat := 'HH:MM:SS';
T1 := StrToTime('00:00:00');
T2 := IncTime(T1, -1, 5, -10, 0);
S := TimeToStr(T2);
end;
```

Routine IncYear

Declaration: function IncYear(ADate: TDateTime; Delta: Integer): TDateTime;

IncYear changes the date ADate by Delta number of years. Delta can be either a positive or negative value.

> IncYear example:

```
var D1, D2: TDateTime;
S: string;
begin
ShortDateFormat := 'dd.mm.yy';
D1 := StrToDate('30.12.1991');
D2 := IncYear(D1, -34);
S := DateToStr(D2);
end;
```

Routine IsLeapYear

Declaration: function IsLeapYear(AYear: Integer): Boolean;

IsLeapYear returns True if the specified year AYear is a leap year.

```
> IsLeapYear example:
```

```
var
Leap: Boolean;
begin
Leap := IsLeapYear(1995);
{ here Leap = False }
end;
```

Routine LastDayOfPrevMonth

Declaration: function LastDayOfPrevMonth: TDateTime;

Returns the last day of the previous month relative to the current system date.

> LastDayOfPrevMonth example:

Last := LastDayOfPrevMonth;

Routine MonthsBetween

Declaration: function MonthsBetween(Date1, Date2: TDateTime): Double;

Count months between Date1 and Date2.

> MonthsBetween example:

Res := MonthsBetween(Date1, Date2);

Routine StrToDateDef

Declaration: function StrToDateDef(const S: string; Default: TDateTime): TDateTime;

StrToDateDef converts a string to a date format. The order for month, day, and year is determined by the ShortDateFormat global variable. If the given string does not contain a valid date, StrToDateDef returns the date passed in Default.

> StrToDateDef example:

Field1.AsDateTime := StrToDateDef(Edit1.Text, SysUtils.Date);

Routine StrToDateFmt

Declaration: function StrToDateFmt(const DateFormat, S: string): TDateTime;

StrToDateFmt converts a string to a date format. The order for month, day, and year is determined by the DateFormat parameter. If the given string does not contain a valid date, an EConvertError exception is raised.

> StrToDateFmt example:

Field1.AsDateTime := StrToDateFmt(DefDateFormat, Edit1.Text);

Routine StrToDateFmtDef

Declaration: function StrToDateFmtDef(const DateFormat, S: string; Default: TDateTime): TDateTime;

StrToDateFmtDef converts a string to a date format. The order for month, day, and year is determined by the DateFormat parameter.

If the given string does not contain a valid date, StrToDateFmtDef returns the date passed in Default.

> StrToDateFmtDef example:

Field1.AsDateTime := StrToDateFmtDef(DefDateFormat, Edit1.Text, SysUtils.Date);

Routine ValidDate

Declaration: function ValidDate(ADate: TDateTime): Boolean;

ValidDate verifies that the specified date ADate is a valid date.

ValidDate example:

if ValidDate(ADate) then
begin
 { deal with ADate }
end;

UNIT FileUtil Routine ClearDir CopyFile DeleteFiles DirExists <u>FileDateTime</u> FileLock FileUnlock **GetFileSize** HasAttr LongToShortFileName LongToShortPath MoveFile NormalDir ShortToLongFileName ShortToLongPath ValidFileName

Routine ClearDir

Declaration: function ClearDir(const Path: string; Delete: Boolean): Boolean;

The ClearDir function erases all files from directory named by Path from the disk. When Delete parameter is True the directory Path also will be erased. If fiels or path cannot be deleted or does not exist, the function returns False.

ClearDir example:

if ClearDir('c:\temp', True) then ShowMessage('Directory C:\TEMP deleted') else ShowMessage('Can"t delete directory C:\TEMP');

Routine CopyFile

Declaration: procedure CopyFile(const FileName, DestName: string; ProgressControl: TControl);

CopyFile copies the file specified by FileName parameter to the new file specified by DestName parameter.

When ProgressControl is not nil, the control specified by ProgressControl (TGauge in 16-bit version or TProgressBar in 32-bit version) will be display the percentage of file copying progress.

CopyFile example:

CopyFile('c:\work\unit1.pas', 'd:\test\unit1.pas', Gauge);

Routine DeleteFiles

Declaration: function DeleteFiles(const FileMask: string): Boolean;

The DeleteFiles function erases all file specified by FileMask from the disk. If files cannot be deleted or does not exist, the function returns False.

DeleteFiles example:

DeleteFiles('work\myproj*.pas');

Routine DirExists

Declaration: function DirExists(Name: string): Boolean;

The DirExists function determines whether the directory specified as the value of the Name parameter exists. If the directory exists, the function returns True. If the directory does not exist, the function returns False. If only a directory name is entered as the value of Name, DirExists searches for the directory within the current directory. If a full path name is entered, DirExists searches for the directory along the designated path.

DirExists example:

This example uses an edit box, a label, and a button on a form. When the user enters a directory name in the edit box and clicks the button, whether or not the directory exists is reported in the caption of the label:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    if DirExists(Edit1.Text) then
       Label1.Caption := Edit1.Text + ' exists'
    else
       Label1.Caption := Edit1.Text + ' does not exist';
end;
```

Routine FileDateTime

Declaration: function FileDateTime(const FileName: string): TDateTime;

FileDateTime returns the date-and-time of the specified file.

FileDateTime example:

D1 := FileDateTime('unit1.pas');

Routine FileLock

Declaration: function FileLock(Handle: Integer; Offset, LockSize: Longint): Integer;

The FileLock function locks a region in an open file. Locking a region prevents other processes from accessing the region. Handle identifies the file with a region to be locked.

Offset specifies the starting byte offset in the file where the lock should begin.

LockSize specifies the length of the byte range to be locked.

FileLock example:

```
var FileHandle : Integer;
begin
  FileHandle := FileOpen(FileName, fmOpenWrite or fmShareDenyNone);
  if FileHandle > 0 then
  try
     { valid file handle }
    if FileLock(FileHandle, 0, 1024) = 0 then
    try
       { first 1024 bytes of opened file is locked }
    finally
       FileUnlock(FileHandle, 0, 1024);
    end;
  finally
    FileClose(FileHandle);
  end else
  {Open error: FileHandle = negative DOS error code};
end;
```

Routine FileUnlock

Declaration: function FileUnlock(Handle: Integer; Offset, LockSize: Longint): Integer;

The FileUnlock function unlocks a region in an open file. Unlocking a region enables other processes to access the region. Handle identifies a file that contains a region locked with FileLock.

Offset specifies the starting byte offset in the file where the locked region begins. LockSize specifies the length of the byte range to be unlocked.

FileUnlock example:

```
var
  FileHandle : Integer;
begin
  FileHandle := FileOpen(FileName, fmOpenWrite or fmShareDenyNone);
  if FileHandle > 0 then
  try
     { valid file handle }
    if FileLock(FileHandle, 0, 1024) = 0 then
    try
       { first 1024 bytes of opened file is locked }
     finally
       FileUnlock(FileHandle, 0, 1024);
    end;
  finally
     FileClose(FileHandle);
  end else
     {Open error: FileHandle = negative DOS error code};
end;
```

Routine GetFileSize

Declaration: function GetFileSize(const FileName: string): Longint;

GetFileSize returns the size of a file, specified by FileName parameter, in bytes.

GetFileSize example:

FSize := GetFileSize('c:\utils\arj.exe');

Routine HasAttr

Declaration: function HasAttr(const FileName: string; Attr: Integer): Boolean;

HasAttr returns True if specified file has attributes Attr.

HasAttr example: if HasAttr('c:\my_file.exe', faReadOnly) then raise Exception.Create('Can't delete read-only file');

Routine LongToShortFileName

Declaration: function LongToShortFileName(const LongName: string): string;

Use this procedure to obtain the short version of the file name specified by LongName parameter.

LongToShortFileName example:

FileName := LongToShortFileName(FileName);

Routine LongToShortPath

Declaration: function LongToShortPath(const LongName: string): string;

Use this procedure to obtain the short version of the directory name specified by LongName parameter.

LongToShortPath example:

DirectoryName := LongToShortPath(DirectoryName);

Routine MoveFile

Declaration: procedure MoveFile(const FileName, DestName: TFileName);

Moves or renames the file passed in FileName to the directory specified as part of DestName parameter.

Tries to just rename the file. If that fails, try to copy the file and delete the original.

Raises an exception if the source file is read-only, and therefore cannot be deleted/moved/renamed.

MoveFile example:

MoveFile(FileName, ChangeFileExt(FileName, '.BAK'));

Routine NormalDir

Declaration: function NormalDir(const DirName: string): string;

Add a default backslash '/' to the end of a directory name DirName. If DirName already has '/' symbol at the end, this function does nothing.

NormalDir example:

{...}
Result := NormalDir(DirName) + ExtractFileName(MyFileName);
{...}

Routine ShortToLongFileName

Declaration: function ShortToLongFileName(const ShortName: string): string;

Use this procedure to obtain the long version of the file name specified by ShortName parameter.

ShortToLongFileName example:

FileName := ShortToLongFileName(FileName);

Routine ShortToLongPath

Declaration: function ShortToLongPath(const ShortName: string): string;

Use this procedure to obtain the long version of the directory name specified by ShortName parameter.

ShortToLongPath example:

DirectoryName := ShortToLongPath(DirectoryName);

Routine ValidFileName

Declaration: function ValidFileName(const FileName: string): Boolean;

ValidFileName indicates whether FileName string refers to a valid name of physical file.

ValidFileName example:

if not ValidFileName(S) then
 ShowMessage('Invalid file name');

Unit <u>FileUtil</u> Routine <u>ClearDir</u> CopyFile **DeleteFiles** DirExists <u>FileDateTime</u> <u>FileLock</u> FileUnlock <u>GetFileSize</u> <u>HasAttr</u> LongToShortFileName LongToShortPath MoveFile NormalDir <u>ShortToLongFileName</u> <u>ShortToLongPath</u> ValidFileName

UNIT MaxMin

Routines: <u>Max</u> <u>MaxFloat</u> <u>MaxInteger</u> <u>MinFloat</u> <u>MinInteger</u> <u>MinOf</u> <u>SwapInt</u> <u>SwapLong</u>

Routine Max

Declaration: function Max(A, B: Longint): Longint;

The Max function returns the largest of two longint values A and B.

Max example: I1 := Max(I1, I2);

Routine MaxFloat

Declaration: function MaxFloat(const Values: array of Extended): Extended;

The MaxFloat function returns the largest signed value in the Values array.

MaxFloat example: MaxVal := MaxFloat([MaxVal, -1, 12.2, 3.36, 12.438]);

Routine MaxInteger

Declaration: function MaxInteger(const Values: array of Longint): Longint;

The MaxInteger function returns the greatest longint value in the Values array.

MaxInteger example: MaxVal := MaxInteger([MaxVal, 1, 12, 36, 43]);

Routine MaxOf

Declaration: function MaxOf(const Values: array of Variant): Variant; The MaxOf function returns the greatest value in the Values array.

MaxOf example: MaxVal := MaxOf([MaxVal, -1, 12.2, 3.36, 12.438]);

Routine Min

Declaration: function Min(A, B: Longint): Longint;

The Max function returns the smallest of two longint values A and B.

Min example: I1 := Min(I1, I2);

Routine MinFloat

Declaration: function MinFloat(const Values: array of Extended): Extended;

The MinFloat function returns the smallest signed value in the Values array.

MinFloat example: MinVal := MinFloat([MinVal, -1, 12.2, 3.36, 12.438]);

Routine MinInteger

Declaration: function MinInteger(const Values: array of Longint): Longint;

The MinInteger function returns the smallest longint value in the Values array.

MinInteger example: MinVal := MinInteger([MinVal, 1, 12, 36, 43]);

Routine MinOf

Declaration: function MinOf(const Values: array of Variant): Variant;

The MinOf function returns the smallest value in the Values array.

MinOf example: MinVal := MinOf([MinVal, -1, 12.2, 3.36, 12.438]);

Routine SwapInt

Declaration: procedure SwapInt(var Int1, Int2: Integer);

Exchange the values in two integers Int1 and Int2.

SwapInt example: SwapInt(I1, I2);

Routine SwapLong

Declaration: procedure SwapLong(var Int1, Int2: Longint);

Exchange the values in two long integers Int1 and Int2.

SwapLong example: SwapLong(I1, I2);

Control Name	Unit	Class
Huge List.	<u>ObjStr</u>	THugeList

Description:

Class THugeList is 16-bit class similar to TList, but the indexes are of type LongInt, rather than Integer. The size of the list is limited only by the amount of virtual memory that can be allocated from Windows. Each item is a Pointer. THugeList never dereferences this pointer, and it can be any value you want it to be. When items are deleted from the list and when the list is destroyed, these items are not freed or dereferenced.

If you know that your indexes will be smaller than 16K, then use TList, not THugeList, to obtain better performance. Note that THugeList is not fully optimized for performance.

NOTE. In 32-bit version the THugeList class is a stub and declared as THugeList = TList;

Class TObjectStrings

Class TObjectStrings is similar to TStringList, but TStringList never dereferences the pointer, stored in the list as Objects property.

Unlike the TStrings and TStringList, when items are deleted from the TObjectStrings list and when the list is destroyed, these items are freed and dereferenced. By default, the DestroyObject virtual method calls the OnDestroyObject event handler, which calls TObject.Free to free and dereference the object.

You can change this behavior by specifying what processing you want to occur in the OnDestroyObject event handler.

Event OnDestroyObject

Declaration: OnDestroyObject: TDestroyEvent;

The OnDestroyObject event occurs when a object stored in the list are freed. By default, the DestroyObject method calls the OnDestroyObject event handler, which calls TObject.Free to free and dereference the object. You can change this behavior by specifying what processing you want to occur in the OnDestroyObject event handler.

Method Remove

Declaration: procedure Remove(Index: Integer);

Removes the item given by Index from the list. Remove does not destroy of item. If you need to delete and destroy of an item, call Delete method.

Const MaxHugeListSize

Declaration: MaxHugeListSize = MaxLongint div SizeOf(Pointer);;

The MaxHugeListSize constant represents the largest number of items a THugeList object can contain.

Type TDestroyEvent Declaration: TDestroyEvent = procedure(Sender, AObject: TObject) of object;

TDestroyEvent is the type of the OnDestroyObject event of the TObjectStrings class.

Unit <u>ObjStr</u>

Class TObjectStrings

Event OnDestroyObject

Method Remove

Constant <u>MaxHugeListSize</u>

Type <u>TDestroyEvent</u>

UNIT Ole2Auto

Class <u>TOleController</u>

Property Locale OleObject

Method **AssignIDispatch CallFunction CallFunctionByID CallFunctionByIDsNamedParams CallFunctionNamedParams CallFunctionNoParams CallFunctionNoParamsByID CallProcedure CallProcedureByID CallProcedureByIDsNamedParams CallProcedureNamedParams CallProcedureNoParams CallProcedureNoParamsByID CreateObject GetProperty GetPropertyByID SetProperty SetPropertyByID**

Routine

CreateLCID

Class TOleController

This class allows you accesses OLE Automation objects. OLE Automation controllers are clients that use the OLE IDispatch interface to access the OLE server objects that implement this interface. The controller must first create the object by calling the CreateObject method, then call the object's interface methods by using the CallFunction, CallFunctionNamedParams, CallProcedure, CallProcedureNamedParams etc methods.

In 32-bit Delphi versions you can use internal Delphi OLE Automation controller instead, but Delphi controller not supports the object's interface methods that named with '\$' symbols, i.e. "GetText\$" (in Word Basic). This class allows you call such methods still.

Property Locale Declaration: Locale: TLCID; Returns the Language ID.

Property OleObject

Declaration: OleObject: Variant;

Read OleObject property to access the OLE object created by the OLE controller as a variant.

Method AssignIDispatch

Declaration: procedure AssignIDispatch(V: Variant); virtual;

procedure AssignDispatch assigns variant *V* to dispatch.

Method CallFunction

Declaration: function CallFunction(const AName: string; const Params: array of const): PVariant;

CallFunction function calls the function name AName with parameters Params.

Method CallFunctionByID

Declaration: function CallFunctionByID(ID: DISPID; const Params: array of const): PVariant;

CallFunctionById calls the function by it's *ID* number and with parameters *Params*

Method CallFunctionByIDsNamedParams

Declaration: function CallFunctionByIDsNamedParams(const IDs: TDispIDList; const Params: array of const; Cnt: Byte): PVariant;

CallFunctionByIdsNamedParams calls the function by it's *IDs* with parameters *Params* with count of parameters *Cnt*.

Method CallFunctionNamedParams

Declaration: function CallFunctionNamedParams(const AName: string; const Params: array of const; const Names: array of string): PVariant;

CallFunctionNamedParams calls a function with Parameters *Params* and *Names*.

Method CallFunctionNoParams

Declaration: function CallFunctionNoParams(const AName: string): PVariant;

CallFunctionNoParams calls the function *AName* without parameters.
Method CallFunctionNoParamsByID

Declaration: function CallFunctionNoParamsByID(ID: DISPID): PVariant;

CallFunctionNoParamsById calls a function by it's ID.

Method CallProcedure

Declaration: procedure CallProcedure(const AName: string; const Params: array of const);

CallProcedure calls a procedure AName with parameters Params.

Method CallProcedureByID

Declaration: procedure CallProcedureByID(ID: DISPID; const Params: array of const);

CallProcedureById calls a procedure by it's ID and with parameters Params

Method CallProcedureByIDsNamedParams

Declaration: procedure CallProcedureByIDsNamedParams(const IDs: TDispIDList; const Params: array of const; Cnt: Byte);

CallProcedureByIdsNamedParams calls a procedure by (ID number) *Ids* and with parameters Params and with parameter count *Cnt*.

Method CallProcedureNamedParams

Declaration: procedure CallProcedureNamedParams(const AName: string; const Params: array of const; const Names: array of string);

CallProcedureByIdsNamedParams calls a procedure by (ID number) *Ids* and with parameters Params and with *Names*.

Method CallProcedureNoParams

Declaration: procedure CallProcedureNoParams(const AName: string);

CallProcedureNoParams calls a procedure with No parameters, and function name AName.

Method CallProcedureNoParamsByID

Declaration: procedure CallProcedureNoParamsByID(ID: DISPID);

CallProcedureNoparamsById calls a procedure without any parameters by it's ID (number).

Method CreateObject

Declaration: procedure CreateObject(const ClassName: string);

The CreateObject method creates an OLE automation object of the specified class. This is the way to get a new instance of an OLE automation server object for your automation controller.

Method GetProperty

Declaration: function GetProperty(const AName: string): PVariant;

Many automation objects include properties in their interfaces. Properties represent the state or content of the automation server. This method allows you to refer to automation-object property by its name AName.

Method GetPropertyByID

Declaration: function GetPropertyByID(ID: DISPID): PVariant;

Many automation objects include properties in their interfaces. Properties represent the state or content of the automation server. This method allows you to refer to automation-object property by its identifier ID.

Method SetProperty

Declaration: procedure SetProperty(const AName: string; const Prop: array of const);

Many automation objects include properties in their interfaces. Properties represent the state or content of the automation server. This method allows you to set a value of automation-object property by its name AName.

Method SetPropertyByID

Declaration: procedure SetPropertyByID(ID: DISPID; const Prop: array of const);

Many automation objects include properties in their interfaces. Properties represent the state or content of the automation server. This method allows you to set a value of automation-object property by its identifier ID.

Routine CreateLCID

Declaration: function CreateLCID(PrimaryLangID, SubLangID: Word): TLCID;

CreateLCID example:

Next code fragment sets the language code for English: OleController.Locale := CreateLCID(LANG_ENGLISH, SUBLANG_DEFAULT);

> Index Page | About | Download Creation Date: 4 Feb 1998 | Last Update: 16 Mar 2000

Unit Ole2Auto

Class TOleController

Property Locale OleObject

Method **AssignIDispatch CallFunction CallFunctionByID CallFunctionByIDsNamedParams CallFunctionNamedParams CallFunctionNoParams CallFunctionNoParamsByID CallProcedure CallProcedureByID CallProcedureByIDsNamedParams CallProcedureNamedParams CallProcedureNoParams CallProcedureNoParamsByID CreateObject GetProperty GetPropertyByID SetProperty** SetPropertyByID

Routine CreateLCID

> Index Page | About | Download Creation Date: 4 Feb 1998 | Last Update: 16 Mar 2000

Unit <u>StrUtils</u>

Constants: Digit Chars WaitCursor

Type <u>TFillDirection</u> <u>TVertAlignment</u>

Routine **ActivatePrevInstance ActivateWindow** <u>AllocMemo</u> **AnsiUpperFirstChar AssignBitmapCell CenterControl CenterWindow ChangeBitmapColor CompareMem** CopyParentImage **CreateBitmapFromIcon CreateTwoColorsBrushPattern DefineCursor Delay DialogUnitsToPixelsX DialogUnitsToPixelsY DrawBitmapRectTransparent DrawBitmapTransparent** DrawCellBitmap **DrawCellText DrawInvertFrame FreeMemo** FreeUnusedOLE **GetEnvVar GetMemoSize GradientFillRect HeightOf HugeDec** HugeInc **HugeMove HugeOffset** KillMessage **LoadAniCursor** LoadDLL **MakeBitmap MakeBitmapID** MakeIcon MakeIconID **MakeModuleBitmap**

<u>MakeModuleIcon</u>
<u>MergeForm</u>
<u>MinimizeText</u>
<u>MsgBox</u>
NotImplemented
PaintInverseRect
<u>PixelsToDialogUnitsX</u>
<u>PixelsToDialogUnitsY</u>
<u>PointInPolyRgn</u>
<u>PointInRect</u>
<u>RegisterServer</u>
<u>ResourceNotFound</u>
<u>ShadeRect</u>
<u>SplitCommandLine</u>
<u>StartWait</u>
<u>StopWait</u>
<u>WidthOf</u>
<u>Win32Check</u>

Const DigitChars

Declaration: DigitChars = ['0'..'9'];;

Digits constant.

Type TCharSet

Declaration: TCharSet = set of Char;

TCharSet is used to specify symbols arrays in procedures and functions of StrUtils unit.

Routine AddChar

Declaration: function AddChar(C: Char; const S: string; N: Integer): string;

AddChar return a string left-padded to length N with characters C.

AddChar example: S := 'Let It Be'; S := AddChar(' ', S, 10); { here S = ' Let It Be' }

Routine AddCharR

Declaration: function AddCharR(C: Char; const S: string; N: Integer): string;

AddCharR return a string right-padded to length N with characters C.

AddCharR example:

S := 'Let It Be'; S := AddChar(' ', S, 10); { here S = 'Let It Be ' }

Routine AnsiProperCase

Declaration: function AnsiProperCase(const S: string; const WordDelims: TCharSet): string;

Returns string, with the first letter of each word in uppercase, all other letters in lowercase. Words are delimited by WordDelims.

AnsiProperCase example:

S := AnsiProperCase('let him go'); { here S = 'Let Him Go' }

Routine CenterStr

Declaration: function CenterStr(const S: string; Len: Integer):string;

CenterStr centers the characters in the string based upon the Len specified.

CenterStr example:

S := 'Let It Be'; S := CenterStr(S, 13); { here S = ' Let It Be ' }

Routine CompStr

Declaration: function CompStr(const S1, S2: string): Integer;

CompStr compares S1 to S2, with case-sensitivity. The return value is -1 if S1 < S2, 0 if S1 = S2, or 1 if S1 > S2.

CompStr example:

```
var I: Integer;
begin
I := CompStr(S1, S2);
if I = 0 then
begin
\{ S1 = S2 \}
end else
if I < 0 then
begin
\{ S1 < S2 \}
end else begin
\{ S1 > S2 \}
```

end; end;

Routine Copy2Space

Declaration: function Copy2Space(const S: string): string;

Copy2Space returns a substring of a string S from begining to first white space.

Copy2Space example:

S := 'Let it be'; S := Copy2Space(S); { here S = 'Let' }

Routine Copy2SpaceDel

Declaration: function Copy2SpaceDel(var S: string): string;

Copy2SpaceDel returns a substring of a string S from begining to first white space and removes this substring from S.

Copy2SpaceDel example:

S := 'Let it be'; S1 := Copy2SpaceDel(S); { here S1 = 'Let', S = 'it be' }

Routine Copy2Symb

Declaration: function Copy2Symb(const S: string; Symb: Char): string;

Copy2Symb returns a substring of a string S from begining to first character Symb.

Copy2Symb example:

S := 'Let-it-be'; S := Copy2Symb(S, '-'); { here S = 'Let' }

Routine Copy2SymbDel

Declaration: function Copy2SymbDel(var S: string; Symb: Char): string;

Copy2SymbDel returns a substring of a string S from begining to first character Symb and removes this substring from S.

Copy2SymbDel example:

S := 'Let-it-be'; S1 := Copy2SymbDel(S, '-'); { here S1 = 'Let', S = 'it-be' }

Routine Dec2Hex

Declaration: function Dec2Hex(N: Longint; A: Byte): string;

Dec2Hex converts the given value to a hexadecimal string representation with the minimum number of digits (A) specified.

Dec2Hex example:

S := Dec2Hex(10, 2); { here S = '0A' }

Routine Dec2Numb

Declaration: function Dec2Numb(N: Longint; A, B: Byte): string;

Dec2Numb converts the given value to a string representation with the base equal to B and with the minimum number of digits (A) specified.

Dec2Numb example:

S := Dec2Numb(10, 8, 2); { here S = '00001010' }

Routine DelBSpace

Declaration: function DelBSpace(const S: string): string;

DelBSpace trims leading spaces from the given string.

DelBSpace example:

```
var S: string;
begin
S := ' Let It Be';
S := DelSpace(S);
ShowMessage(S); { here S = 'Let It Be' }
end;
```

Routine DelChars

Declaration: function DelChars(const S: string; Chr: Char): string;

DelChars return a string with all Chr characters removed.

DelChars example: var S: string;

```
begin
    S := 'Let It Be';
    S := DelChars(S, 'e');
    ShowMessage(S); { here S = 'Lt It B' }
end;
```

Routine DelESpace

Declaration: function DelESpace(const S: string): string;

DelESpace trims trailing spaces from the given string.

DelESpace example:

```
var S: string;
begin
    S := 'Let It Be ';
    S := DelSpace(S);
    ShowMessage(S); { here S = 'Let It Be' }
end;
```

Routine DelRSpace

Declaration: function DelRSpace(const S: string): string;

DelRSpace trims leading and trailing spaces from the given string.

DelRSpace example:

```
var S: string;
begin
    S := ' Let It Be ';
    S := DelSpace(S);
    ShowMessage(S); { here S = 'Let It Be' }
end;
```

Routine DelSpace

Declaration: function DelSpace(const S: string): string;

DelSpace return a string with all white spaces removed.

> DelSpace example:

var S: string; begin S := 'Let It Be'; S := DelSpace(S);

```
ShowMessage(S); { here S = 'LetItBe' } end;
```

Routine DelSpace1

Declaration: function DelSpace1(const S: string): string;

DelSpace1 return a string with all non-single white spaces removed.

> DelSpace1 example:

var S: string;

```
begin
    S := 'Let It Be';
    S := DelSpace1(S);
    ShowMessage(S); { here S = 'Let It Be' }
end;
```

Routine ExtractDelimited

Declaration: function ExtractDelimited(N: Integer; const S: string; const Delims: TCharSet): string;

ExtractWord, ExtractWordPos and ExtractDelimited given a set of word delimiters, return the N'th word in S.

ExtractDelimited example:

S := 'Let him go'; S1 := ExtractDelimited(2, S, [' ']); { here S1 = " }

Routine ExtractSubstr

Declaration: function ExtractSubstr(const S: string; var Pos: Integer; const Delims: TCharSet): string;

ExtractSubstr given a set of word delimiters, return the substring from S, that started from position Pos.

ExtractSubstr example:

```
var
Pos: Integer; List: TStringList; S: string;
begin
{ ... }
Pos := 1;
while Pos <= Length(S) do
List.Add(ExtractFieldName(S, Pos, [',',';']));
{ ... }
end;
```

Routine ExtractWord

Declaration: function ExtractWord(N: Integer; const S: string; const WordDelims: TCharSet): string;

ExtractWord, ExtractWordPos and ExtractDelimited given a set of word delimiters, return the N'th word in S.

> ExtractWord example:

S := 'Let him go'; S1 := ExtractWord(2, S, [' ']); { here S1 = 'him' }

Routine ExtractWordPos

Declaration: function ExtractWordPos(N: Integer; const S: string; const WordDelims: TCharSet; var Pos: Integer): string;

ExtractWord, ExtractWordPos and ExtractDelimited given a set of word delimiters, return the N'th word in S.

> ExtractWordPos example: S := 'Let him go'; S1 := ExtractWordPos(2, S, [' '], P); { here S1 = 'him', P = 6 }

Routine FindPart

Declaration: function FindPart(const HelpWilds, InputStr: string): Integer;

FindPart compares a string with '?' and another, returns the position of HelpWilds in InputStr.

FindPart example:

var P: Integer; Wild: string; begin Wild := '?im'; P := FindPart(Wild, 'Let him go'); { here P = 5 } end;

Routine Hex2Dec

Declaration: function Hex2Dec(const S: string): Longint;

Hex2Dec converts the given hexadecimal string to the corresponding integer value.

Hex2Dec example:

I := Hex2Dec('1A');

{ here I = 26 }

Routine IntToRoman

Declaration: function IntToRoman(Value: Longint): string;

IntToRoman converts the given value to a roman numeric string representation.

IntToRoman example:

S := IntToRoman(1996);

Routine IsEmptyStr

Declaration: function IsEmptyStr(const S: string; const EmptyChars: TCharSet): Boolean;

EmptyStr returns True if the given string contains only character from the EmptyChars.

IsEmptyStr example:

if IsEmptyStr(S, [#0,' ']) then begin ShowMessage('S contains only while spaces'); end;

Routine IsWild

Declaration: function IsWild(InputStr, Wilds: string; IgnoreCase: Boolean): Boolean;

IsWild compare InputString with WildCard string and return True if corresponds.

IsWild example: There are possible masks and corresponding strings:

* : >= 0 letters; *A : words with >= 1 letters and A at the end; A*A : words with >= 2 letters and A at the begin and end; A* : words with >= 1 letters and A at the begin; ? : one letter.

Routine IsWordPresent

Declaration: function IsWordPresent(const W, S: string; const WordDelims: TCharSet): Boolean;

IsWordPresent given a set of word delimiters, return True if word W is present in string S.

IsWordPresent example:

if IsWordPresent('him', 'Let him go', [' ', ',']) then

ShowMessage('String contsins word "him"');

Routine LeftStr

Declaration: function LeftStr(const S: string; N: Integer): string;

LeftStr return a string right-padded to length N with blanks.

LeftStr example:

S := 'Let It Be'; S := LeftStr(S, 10); { here S = 'Let It Be ' }

Routine MakeStr

Declaration: function MakeStr(C: Char; N: Integer): string;

MakeStr return a string of length N filled with character C.

MakeStr example: S := MakeStr(' ', 10);

Routine MS Declaration: function MS(C: Char; N: Integer): string;

MS return a string of length N filled with character C.

MS example: S := MS(' ', 10);

Routine NPos

Declaration: function NPos(const C: string; S: string; N: Integer): Integer;

NPos searches for a N-th position of substring C in a given string S.

> NPos example:

```
var
   S: string; I: Integer;
begin
   S := 'Let It Be';
   I := NPos(' ', S, 2);
   { I = 7 }
end;
```

Routine Numb2Dec

Declaration: function Numb2Dec(S: string; B: Byte): Longint;

Numb2Dec converts the given B-based numeric string to the corresponding integer value.

Numb2Dec example:

I := Numb2Dec('1010', 2); { here I = 10 }

Routine Numb2USA

Declaration: function Numb2USA(const S: string): string;

Numb2USA converts numeric string S to USA-format.

Numb2USA example:

S := Numb2USA('12365412'); { here S = '12,365,412' }

Routine OemToAnsiStr

Declaration: function OemToAnsiStr(const OemStr: string): string;

OemToAnsiStr translates a string from the OEM character set into the Windows character set.

> OemToAnsiStr example: S := OemToAnsiStr(S);

Routine ReplaceStr

Declaration: function ReplaceStr(const S, Srch, Replace: string): string;

Returns string with every occurrence of Srch string replaced with Replace string.

ReplaceStr example:

S := ReplaceStr(ReplaceStr(Condition, '*', '%'), '?', '_');

Routine RightStr

Declaration: function RightStr(const S: string; N: Integer): string;

RightStr return a string left-padded to length N with blanks.

RightStr example: S := 'Let It Be'; S := RightStr(S, 10); { here S = ' Let It Be' }

Routine RomanToInt

Declaration: function RomanToInt(const S: string): Longint;

RomanToInt converts the given string to an integer value. If the string doesn't contain a valid roman numeric value, the 0 value is returned.

RomanToInt example:

I := RomanToInt('MCMXVI');

Routine StrToOem

Declaration: function StrToOem(const AnsiStr: string): string;

StrToOem translates a string from the Windows character set into the OEM character set.

StrToOem example:

S := StrToOem(S);

Routine Tab2Space

Declaration: function Tab2Space(const S: string; Numb: Byte): string;

Tab2Space converts any tabulation character in the given string to the Numb spaces characters.

Tab2Space example:

S := Tab2Space(S, 8);

Routine WordCount

Declaration: function WordCount(const S: string; const WordDelims: TCharSet): Integer;

WordCount given a set of word delimiters, return number of words in S.

WordCount example: I := WordCount('Let him go', [' ', ',']); { here I = 3 }

Routine WordPosition

Declaration: function WordPosition(const N: Integer; const S: string; const WordDelims: TCharSet): Integer;

Given a set of word delimiters, return start position of N'th word in S.

WordPosition example: I := WordPosition(2, 'Let him go', [' ', ',']); { here I = 5 } Index Page | About | Download Creation Date: 4 Feb 1998 | Last Update: 16 Mar 2000 Unit <u>StrUtils</u>

Constants: Digit Chars WaitCursor

Type <u>TFillDirection</u> <u>TVertAlignment</u>

Routine **ActivatePrevInstance ActivateWindow** <u>AllocMemo</u> **AnsiUpperFirstChar AssignBitmapCell CenterControl CenterWindow ChangeBitmapColor CompareMem** CopyParentImage **CreateBitmapFromIcon CreateTwoColorsBrushPattern DefineCursor Delay DialogUnitsToPixelsX DialogUnitsToPixelsY DrawBitmapRectTransparent DrawBitmapTransparent** DrawCellBitmap **DrawCellText DrawInvertFrame FreeMemo** FreeUnusedOLE **GetEnvVar GetMemoSize GradientFillRect HeightOf HugeDec** HugeInc **HugeMove HugeOffset** KillMessage **LoadAniCursor** LoadDLL **MakeBitmap MakeBitmapID** MakeIcon MakeIconID **MakeModuleBitmap**

<u>MakeModuleIcon</u>
<u>MergeForm</u>
<u>MinimizeText</u>
<u>MsgBox</u>
<u>NotImplemented</u>
PaintInverseRect
<u>PixelsToDialogUnitsX</u>
<u> PixelsToDialogUnitsY</u>
<u>PointInPolyRgn</u>
<u>PointInRect</u>
<u>RegisterServer</u>
<u>ResourceNotFound</u>
<u>ShadeRect</u>
<u>SplitCommandLine</u>
<u>StartWait</u>
<u>StopWait</u>
<u>WidthOf</u>
<u>Win32Check</u>

Index Page | About | Download Creation Date: 4 Feb 1998 | Last Update: 16 Mar 2000 UNIT SplshWnd

Constants: <u>SplashStayOnTop</u>

Types: <u>TSplashWindow</u>

Routine: <u>ShowSplashWindow</u>

Const SplashStayOnTop

Declaration: SplashStayOnTop: Boolean = True;;

This global variable determines whether or not the form created by ShowSplashWindow function will be have FormStyle property equal to fsStayOnTop. The default setting is True. Set SplashStayOnTop to False if you want to create splash screen with form style equal to fsNormal.

Type TSplashWindow

Declaration: TSplashWindow = class(TForm);

This calss is intended for creating splash screens. Splash screens are simply forms that are presented to users while the application is performing time-consuming operations or displayed as soon as your application program is executed.

To create and display splash screen use ShowSplashWindow function.

Routine ShowSplashWindow

Declaration: function ShowSplashWindow(Graphic: TGraphic; const MsgText: string; Animate: Boolean; AlignForm: TForm): TSplashWindow;

ShowSplashWindow creates and displays splash form. Splash forms are simply forms that are presented to users while the application is performing time-consuming operations or displayed as soon as your application program is executed. While the splash screen is being displayed, you can perform other operations in your program.

Graphic parameter specified a graphic to displaing in left part of splash screen. It may be icon, bitmap or "animated" bitmap. When Animate parameter is True and Graphic is TBitmap object, then TAnimatedImage component will be created to display animation in splash screen. Of course you must call Application.ProcessMessages method while the splash screen is being displayed to allow animation.

MsgText parameter specified the text string that will be displayed on the splash screen. AlignForm is a form withih the splash screen will be centered. Function returns a TSplashForm object. Don't forget to destroy splash screen after displaying by calling its Free destructor.

ShowSplashWindow example:

var Splash: TForm; begin Splash := ShowSplashWindow(Application.Icon, 'Loading... Please wait.', False, Application.MainForm); try { time-consuming operations } finally Splash.Free; end; end;

> Index Page | About | Download Creation Date: 4 Feb 1998 | Last Update: 16 Mar 2000

UNIT VCLUtils

Constant <u>WaitCursor</u>

Type <u>TFillDirection</u> <u>TVertAlignment</u>

Routines **ActivatePrevInstance ActivateWindow** <u>AllocMemo</u> **AnsiUpperFirstChar** <u>AssignBitmapCell</u> **CenterControl** CenterWindow **ChangeBitmapColor CompareMem CopyParentImage** CreateBitmapFromIcon **CreateTwoColorsBrushPattern DefineCursor Delay DialogUnitsToPixelsX DialogUnitsToPixelsY DrawBitmapRectTransparent DrawBitmapTransparent DrawCellBitmap DrawCellText DrawInvertFrame FreeMemo FreeUnusedOLE GetEnvVar GetMemoSize** GradientFillRect **HeightOf HugeDec** HugeInc **HugeMove HugeOffset KillMessage** LoadAniCursor **LoadDLL** <u>MakeBitMap</u> **MakeBitMapId MakeIcon MakeIconId** MakeModuleBitMap **MakeModuleIcon MergeForm**

MinimizeText MsgBox NotImplemented PaintInverseRect PixelsToDialogUnitsX PixelsToDialogUnitsY PointInPolyRgn PointInRect RegisterServer ResourceNotFound **ShadeRect SplitCommandLine** <u>StartWait</u> **StopWait** WidthOf **LoadAniCursor LoadDLL MakeBitmap MakeBitmapID MakeIcon MakeIconID MakeModuleBitmap MakeModuleIcon MergeForm** MinimizeText **MsgBox NotImplemented** PaintInverseRect **PixelsToDialogUnitsX PixelsToDialogUnitsY PointInPolyRgn PointInRect RegisterServer ResourceNotFound ShadeRect SplitCommandLine StartWait StopWait** WidthOf Win32Check

Const WaitCursor

Declaration: WaitCursor: TCursor = crHourGlass;

This defines the default wait cursor as the hour glass.

Type TFillDirection

Declaration: TFillDirection = (fdTopToBottom, fdBottomToTop, fdLeftToRight, fdRightToLeft);

TFillDirection defines the possible values of the Direction parameter for GradientFillRect procedure.

Type TVertAlignment

Declaration: TVertAlignment = (vaTopJustify, vaCenter, vaBottomJustify);

TVertAlignment is a set of values for the VertAlign parameter of the DrawCellText procedure.

Routine ActivatePrevInstance

Declaration: function ActivatePrevInstance(const MainFormClass, ATitle: string): Boolean;

ActivatePrevInstance function allows you to stop your application from being running more than one instance under Windows. It causes the application to detect another instance at startup and when found, terminates the second instance and tries to activate the original instance of your application. ActivatePrevInstance find the top-level window whose class name and window name match the specified strings and, when found, activate this window and returns True. Typically, you can use name of class of main application form as MainFormClass parameter. ATitle parameter specifies the window name (the window's title). If this parameter is empty string, all window names match. Also you can use 'TApplication' as MainFormClass parameter and Application.Title as ATitle parameter. It's recomended however to use the main form class name so it can be more unique for Windows.

You can call this function in project source (in .DPR-file) of your application.

ActivatePrevInstance example:

You can call ActivatePrevInstance in project source file (DPR) to stop your application from being running more than one instance under Windows:

begin
 if ActivatePrevInstance(TRxMainProjectForm.ClassName, ") then
 Exit;
 Application.CreateForm(TRxMainProjectForm, RxMainProjectForm);
 { create other forms }
 Application.Run;
end;

Routine ActivateWindow

Declaration: procedure ActivateWindow(Wnd: HWnd);

The ActivateWindow function activates a window and forces the thread that created the window into the foreground. An application should call ActivateWindow if it wants to put itself into the foreground.

If window specified by Wnd handle is hidden, then ActivateWindow displays the window. If the window is minimized ActivateWindow restores it to its original size and position.

ActivateWindow example: ActivateWindow(Wnd);

Routine AllocMemo

Declaration: function AllocMemo(Size: Longint): Pointer;

AllocMemo allocates a block of the given Size on the heap. Each byte in the allocated buffer is set to zero. To dispose the buffer, use the FreeMemo procedure.

AllocMemo example:

```
var
P: Pointer;
begin
P := AllocMemo(128000);
try
{ use memory }
finally
FreeMemo(P);
end;
end;
```

Routine AnsiUpperFirstChar

Declaration: function AnsiUpperFirstChar(const S: string): string;

This returns the string, processed with the first letter in uppercase, all other letters in lowercase, using ANSI character set.

AnsiUpperFirstChar example: ClientName := AnsiUpperFirstChar(ClientName);

Routine AssignBitmapCell

Declaration: procedure AssignBitmapCell(Source: TGraphic; Dest: TBitmap; Cols, Rows, Index: Integer);

AssignBitmapCell example:

Next code fragment loads standard Windows "box" images, cut cell from it and assign selected bitmap to Glyph property of TSpeedButton component:

```
var
Bmp, Checks: TBitmap;
begin
Bmp := TBitmap.Create;
Checks := MakeModuleBitmap(0, PChar(32759));
try
{...}
AssignBitmapCell(Checks, Bmp, 4, 3, 1);
SpeedButton1.Glyph := Bmp;
{...}
finally
Bmp.Free;
Checks.Free;
```

end; end;

Routine CenterControl

Declaration: procedure CenterControl(Control: TControl);

This procedure centers the given window control over its parent. If the control is a form, it is centered over the screen. If the control is a MDI-child form, it is centered over the main application MDI-form. It ensures that the window is entirely within the visible screen.

CenterControl example:

procedure TForm1.FormResize(Sender: TObject); begin CenterControl(Memo1); end;

Routine CenterWindow

Declaration: procedure CenterWindow(Wnd: HWnd);

This function centers the given window Wnd over the screen. It ensures that the window is entirely within the visible screen.

CenterWindow example:

CenterWindow(Form1.Hanle);

Routine ChangeBitmapColor

Declaration: function ChangeBitmapColor(Bitmap: TBitmap; Color, NewColor: TColor): TBitmap;

The function ChangeBitmapColour changes the *colour* inside the bitmap to the corresponding *NewColor*.

ChangeBitmapColor example:

```
var
Bmp1, Bmp2: TBitmap;
begin
{...}
Bmp2 := ChangeBitmapColor(Bmp1, clOlive, clSilver);
try
Bmp1.Assign(Bmp2);
finally
Bmp2.Free;
end;
end;
```

Routine CompareMem

Declaration: function CompareMem(fpBlock1, fpBlock2: Pointer; Size: Integer): Boolean; assembler;

CompareMem performs a binary compare of Size bytes of memory referenced by fpBlock1 to that of fpBlock2. CompareMem returns True if the memory referenced by fpBlock1 is identical to that of fpBlock2.

CompareMem example:

if CompareMem(fp1, fp2, 1024) then
begin
 { ... }
end;

Routine CopyParentImage

Declaration: procedure CopyParentImage(Control: TControl; Dest: TCanvas);

CopyParentImage copies the image contained within the control to a tCanvas.

> CopyParentImage example:

procedure TMyControl.Paint; begin Canvas.FillRect(Bounds(0, 0, Width, Height)); { copy image from parent and back-level controls } CopyParentImage(Self, Canvas); DrawBitmapTransparent(Canvas, 0, 0, FImage, FImage.TransparentColor); end;

Routine CreateBitmapFromIcon

Declaration: function CreateBitmapFromIcon(Icon: TIcon; BackColor: TColor): TBitmap;

CreateBitMapFromIcon, as it's name implies, creates a bitmap from an icon. The BackColour is important because there is no bitmap equivalent of the colour "transparent".

```
> CreateBitmapFromIcon example:
var
Bmp: TBitmap;
begin
Bmp := CreateBitmapFromIcon(Form1.Icon, clSilver);
try
{ use bitmap }
finally
Bmp.Free;
end;
end;
```

Routine CreateTwoColorsBrushPattern

Declaration: function CreateTwoColorsBrushPattern(Color1, Color2: TColor): TBitmap;

CreateTwoColoursBrushPattern, as it's name implies, creates a b

> CreateTwoColorsBrushPattern example:

```
var
Bmp: TBitmap;
begin
Bmp := CreateTwoColorsBrushPattern(clBtnFace, clWhite);
try
Canvas.Brush.Bitmap := Bmp;
{ ... }
finally
Bmp.Free;
end;
end;
```

Routine DefineCursor

Declaration: function DefineCursor(Instance: THandle; ResID: PChar): TCursor;

DefineCursor function loads the specified cursor resource from the executable (.EXE or .DLL) file associated with an specified instance and assign a unique index. Returns the new index. ResID parameter points to a null-terminated string that contains the name of the cursor resource to be loaded. Alternatively, this parameter can consist of the resource identifier in the low-order word and zero in the high-order word. The MakeIntResource function can also be used to create this value. If the specified resource can not be found, DefineCursor raises an EResNotFound exception. To use DefineCursor, just call it and assign the return value to a component's Cursor property.

In 32-bit version DefineCursor searches for the specified cursor resource in this sequence:

1. Resources of RT_CURSOR (standard Windows cursors) type.

2. Resources of RT_ANICURSOR (animated cursors) type.

> DefineCursor example:

constructor TDrawingComponent.Create(Owner: TComponent);
begin
inherited Create(Owner);
Cursor := DefineCursor(HInstance, 'CUR_MYCURSOR');
end;

Routine Delay

Declaration: procedure Delay(MSecs: Longint);

Delays a specified number (MSecs) of milliseconds.

> Delay example:

StartWait; try Delay(1000); finally StopWait; end;

Routine DialogUnitsToPixelsX

Declaration: function DialogUnitsToPixelsX(DlgUnits: Word): Word;

Convert dialog units to pixels in horizontal direction. The dialog box base units used by Windows to create dialog boxes.

> DialogUnitsToPixelsX example: X := DialogUnitsToPixelsX(X);

Routine DialogUnitsToPixelsY

Declaration: function DialogUnitsToPixelsY(DlgUnits: Word): Word;

Convert dialog units to pixels in vertical direction. The dialog box base units used by Windows to create dialog boxes.

> DialogUnitsToPixelsY example:

```
Y := DialogUnitsToPixelsY(Y);
```

Routine DrawBitmapRectTransparent

Declaration: procedure DrawBitmapRectTransparent(Dest: TCanvas; DstX, DstY: Integer; SrcRect: TRect; Bitmap: TBitmap; TransparentColor: TColor);

DrawBitmapRectTransparent paints the rectangular part (specified by SrcRect parameter) of bitmap specified by the Bitmap parameter on the canvas Dest at the location given by the coordinates (DstX, DstY). TransparentColor determines which color of the bitmap is to be transparent when the bitmap is drawn.

To draw whole bitmap image transparency use DrawBitmapTransparent procedure.

> DrawBitmapRectTransparent example:

var Bmp: TBitmap; begin Bmp := Image1.Picture.Bitmap; DrawBitmapRectTransparent(PaintBox1.Canvas, 10, 10, Rect(0, 0, Bmp.Width div 2, Bmp.Height div 2), Bmp, clOlive); end;

Routine DrawBitmapTransparent

Declaration: procedure DrawBitmapTransparent(Dest: TCanvas; DstX, DstY: Integer; Bitmap: TBitmap; TransparentColor: TColor);

DrawBitmapTransparent paints the bitmap specified by the Bitmap parameter on the canvas Dest at the location given by the coordinates (DstX, DstY). TransparentColor determines which color of the bitmap is to be transparent when the bitmap is drawn. To render only part of bitmap use DrawBitmapRectTransparent procedure.

> DrawBitmapTransparent example:

var Bmp: TBitmap; begin Bmp := Image1.Picture.Bitmap; DrawBitmapTransparent(PaintBox1.Canvas, 10, 10, Bmp, clOlive); end;

Routine DrawCellBitmap

Declaration: procedure DrawCellBitmap(Control: TCustomControl; ACol, ARow: Longint; Bmp: TBitmap; Rect: TRect);

DrawBitmap draws a bitmap *Bmp* onto *Control* (assumed as DrawGrid) with *ACol* and *ARow* as it's specifier. The bitmap occupies the space *Rect*.

> DrawCellBitmap example:

```
procedure TForm1.Grid1DrawCell(Sender: TObject; Col, Row: Longint; Rect: TRect; State:
TGridDrawState);
var
Bmp: TBitmap;
begin
Bmp := GetCellBitmap(Col, Row);
if Bmp <> nil then
DrawCellBitmap(Grid1, Col, Row, Bmp, Rect);
end;
```

Routine DrawCellText

Declaration: procedure DrawCellText(Control: TCustomControl; ACol, ARow: Longint; const S: string; const ARect: TRect; Align: TAlignment; VertAlign: TVertAlignment);

DrawCellText draws the text (*const s : string*) inside the TDrawGrid, with parameter *ACol* (Column) and *ARow* (row) inside *ARect* with specified Horizontal Alignment *Align* and Vertical Alignment *VertAlign*.

DrawCellText example:

```
procedure TForm1.DrawGrid1DrawCell(Sender: TObject; Col, Row: Longint; Rect: TRect; State:
TGridDrawState);
begin
    if (Row >= 0) then
    begin
       DrawCellText(Sender as TDrawGrid, Col, Row,
       GetCellText(Row), Rect, taLeftJustify, vaCenter);
    end;
end;
```

Routine DrawInvertFrame

Declaration: procedure DrawInvertFrame(ScreenRect: TRect; Width: Integer);

DrawInvertFrame draws a rectangular frame in the screen by performing a logical XOR operation on the color values for each pixel of the rectangle's frame. Because DrawInvertFrame uses an XOR function, calling it a second time while specifying the same rectangle removes the inverted rectangle from the screen. ScreenRect parameter contains the logical coordinates of the rectangle in the screen coordinates (relative to the upper-left corner of the screen). Width parameter specifies the width of the frame in pixels.

> DrawInvertFrame example:

var
P: TPoint;
R: TRect;
begin
P := Panel1.ClientToScreen(Point(0, 0));
R := Bounds(P.X, P.Y, Panel1.Width, Panel1.Height);
DrawInvertFrame(R, 3); { Show invert frame }
{ ... }
DrawInvertFrame(R, 3); { Hide invert frame }
end;

Routine FreeMemo

Declaration: procedure FreeMemo(var fpBlock: Pointer);

The FreeMemo procedure disposes of a dynamic variable. fpBlock is a variable of any pointer type previously assigned by the AllocMemo function.

```
> FreeMemo example:
```

```
var
P: Pointer;
begin
P := AllocMemo(128000);
try
{ use memory }
finally
FreeMemo(P);
```

end; end;

Routine FreeUnusedOLE

Declaration: procedure FreeUnusedOLE;

This procedure unloads OLE DLL's in 32-bit Delphi versions. You can unload unused OLE DLL's when you don't use OLE libraries in your application. Be sure that your project does not calling any OLE functions and does not using Variant types before calling this procedure!

> FreeUnusedOLE example:

program Project1; uses Forms, VCLUtils, Main in 'MAIN.PAS' {MainForm};

{\$R *.RES}

begin

FreeUnusedOLE; Application.CreateForm(TMainForm, MainForm); Application.Run; end;

Routine GetEnvVar

Declaration: function GetEnvVar(const VarName: string): string;

Returns a string value of a specified environment variable.

> GetEnvVar example:

uses SysUtils, Dialogs, VCLUtils; var EnvVar: string;

begin

EnvVar := GetEnvVar('TEMP');

ShowMessage(Format('Path of the temporary file is currently: %s', [EnvVar])); end.

Routine GetMemoSize

Declaration: function GetMemoSize(fpBlock: Pointer): Longint;

The GetMemoSize function retrieves the current size, in bytes, of the specified dynamic variable. fpBlock is a variable of any pointer type previously assigned by the AllocMemo function.

> GetMemoSize example:

```
var P: Pointer; S: Longint;
begin
    P := AllocMemo(128000);
    try
        { use memory }
        S := GetMemoSize(P);
        { ... }
    finally
        FreeMemo(P);
end;
end;
```

Routine GradientFillRect

Declaration: procedure GradientFillRect(Canvas: TCanvas; Rect: TRect; BeginColor, EndColor: TColor; Direction: TFillDirection; Colors: Byte);

GradientFillRect procedure displays a rectangle Rect on a Canvas with a color pattern that blends from one choosen color BeginColor to another color EndColor. Colors parameter determines the number of color bands that are painted between the starting and ending colors. The valid range for Colors is 1 to 255. Direction parameter determines whether the gradient is horizontal or vertical.

GradientFillRect example:

procedure TForm1.FormPaint(Sender: TObject); begin GradientFillRect(Canvas, ClientRect, clBlue, clBlack, fdTopToBottom, 255); end; {...} procedure TForm1.FormResize(Sender: TObject); begin Invalidate; end;

Routine HeightOf

Declaration: function HeightOf(R: TRect): Integer;

Retrieves the vertical size (height) of the specified rectangle R in pixels.

HeightOf example:

H := HeightOf(Rect);

Routine HugeDec

Declaration: procedure HugeDec(var HugePtr: Pointer; Amount: Longint);

Decrement a huge pointer.

HugeDec example:

var P: Pointer; L: Longint;
{...}
HugeDec(P, L);
{...}

Routine HugeInc

Declaration: procedure HugeInc(var HugePtr: Pointer; Amount: Longint);

Increment a huge pointer.

HugeInc example:

var
 P: Pointer;
 L: Longint;
 {...}
 HugeInc(P, L);
 {...}

Routine HugeMove

Declaration: procedure HugeMove(Base: Pointer; Dst, Src, Size: Longint);

Copy memory from Src to Dst, copying Size units from Base+Src to Base+Dst. The reason to separate the Base pointer from the Dst and Src indexes is to easily compare the indexes to see what direction to copy, without having to muck about with normalizing segments. Offsets and the size are in longwords, for convenience, but you need to convert them to short words, in case the code is running on a 286 machine, which does not support the movsd instruction, just movsw.

HugeMove example:

```
var
    P: Pointer;
    L: Longint;
begin
    {...}
    L := 1024 * 82; {82 K}
    HugeMove(P, 0, L, L);
    {...}
end:
```

Routine HugeOffset

Declaration: function HugeOffset(HugePtr: Pointer; Amount: Longint): Pointer;

Add an offset to a huge pointer and return the result.

> HugeOffset example: P := HugeOffset(P, L);

Routine KillMessage

Declaration: procedure KillMessage(Wnd: HWnd; Msg: Cardinal);

KillMessage deletes the requested message Msg from the window message queue, but throw back any WM_QUIT messages that PeekMessage standard Windows function may also return.

> KillMessage example: KillMessage(Form1.Handle, Msg.Message);

Routine LoadAniCursor

Declaration: function LoadAniCursor(Instance: THandle; ResID: PChar): HCursor;

The LoadAniCursor function loads the specified animated cursor resource from the executable (.EXE or .DLL) file associated with the specified application instance. Instance parameter identifies an instance of the module whose executable file contains the animated cursor to be loaded. ResID parameter points to a null-terminated string that contains the name of the animated cursor resource (RT_ANICURSOR type) to be loaded. Alternatively, this parameter can consist of the resource identifier in the low-order word and zero in the high-order word. The MakeIntResource function can also be used to create this value. If the function succeeds, the return value is the handle of the newly loaded animated cursor. If the function fails, the return value is 0 (zero).

```
> LoadAniCursor example:
var Cur: HCursor;
begin
   {...}
   Cur := LoadAniCursor(hInstance, 'CUR_COIN');
   if Cur = 0 then
      { error loading cursor }
   else
      Screen.Cursors[200] := Cur;
   {...}
end;
```

Routine LoadDLL

Declaration: function LoadDLL(const LibName: string): THandle;

The LoadDLL function loads the specified library module. Returns the instance handle of the loaded library module if the function is successful. Otherwise, an EOutOfResources exception is raised.

LoadDLL example:

{\$IFDEF WIN32} TUHandle := LoadDLL('TUTIL32.DLL'); {\$ELSE} TUHandle := LoadDLL('TUTILITY.DLL'); {\$ENDIF}

Routine MakeBitmap

Declaration: function MakeBitmap(ResID: PChar): TBitmap;

MakeBitmap instantiates a bitmap object (TBitmap), and loads the specified bitmap resource from an application's executable file. Specify the resource to be loaded as the value of ResID. When you no longer need the bitmap, call it's Destory destructor to free it. If the function fails, the return value is Nil.

MakeBitmap example:

```
var
Bmp: TBitmap;
begin
Bmp := MakeBitmap('MY_BITMAP');
try
{ use bitmap }
finally
Bmp.Free;
end;
end;
```

Routine MakeBitmapID

Declaration: function MakeBitmapID(ResID: Word): TBitmap;

MakeBitmapID instantiates a bitmap object (TBitmap), and loads the specified bitmap resource from an application's executable file. Specify the resource to be loaded as the value of ResID. When you no longer need the bitmap, call it's Destory destructor to free it. If the function fails, the return value is Nil.

MakeBitmapID example:

```
const ResID = 23012;
var Bmp: TBitmap;
begin
  Bmp := MakeBitmapID(ResID);
  try
    { use bitmap }
  finally
    Bmp.Free;
  end;
end;
```

Routine MakeIcon

Declaration: function MakeIcon(ResID: PChar): TIcon;

MakeIcon instantiates an icon object (TIcon), and loads the specified icon resource from an application's executable file. Specify the resource to be loaded as the value of ResID. When you no longer need the icon, call it's Destory destructor to free it. If the function fails, the return value is Nil.

> MakeIcon example:

```
var
Ico: TIcon;
begin
Ico := MakeIcon('MY_ICON');
try
{ use icon }
finally
Ico.Free;
end;
end;
```

Routine MakeIconID

Declaration: function MakeIconID(ResID: Word): TIcon;

MakeIconID instantiates an icon object (TIcon), and loads the specified icon resource from an application's executable file. Specify the resource to be loaded as the value of ResID. When you no longer need the icon, call it's Destory destructor to free it. If the function fails, the return value is Nil.

MakeIconID example:

const ResID = 23012; var Ico: TIcon; begin Ico := MakeIconID(ResID); try { use icon } finally Ico.Free; en d; end;

Routine MakeModuleBitmap

Declaration: function MakeModuleBitmap(Module: THandle; ResID: PChar): TBitmap;

MakeModuleBitmap instantiates a bitmap object (TBitmap), and loads the specified bitmap resource from a module's executable file. Specify the resource to be loaded as the value of ResID which points to a null-terminated string that contains the name of the bitmap resource to be loaded. Alternatively, this parameter can consist of the resource identifier in the low-order word and zero in the high-order word. The MakeIntResource function can be used to create this value. Module parameter identifies the instance of the module whose executable file contains the bitmap to be loaded.

When you no longer need the bitmap, call it's Destory destructor to free it.
An application can use the MakeModuleBitmap function to access the predefined bitmaps used by the Windows API. To do so, the application must set the Module parameter to zero. If the function fails, the return value is Nil.

MakeModuleBitmap example:

var Bmp: TBitmap;

```
begin
   Bmp := MakeModuleBitmap(0, PChar(32759)); { check box image }
   try
      { use bitmap }
   finally
      Bmp.Free;
   end;
end;
```

Routine MakeModuleIcon

Declaration: function MakeModuleIcon(Module: THandle; ResID: PChar): TIcon;

MakeModuleIcon instantiates an icon object (TIcon), and loads the specified icon resource from a module's executable file. Specify the resource to be loaded as the value of ResID which points to a null-terminated string that contains the name of the icon resource to be loaded. Alternatively, this parameter can consist of the resource identifier in the low-order word and zero in the high-order word. The MakeIntResource function can be used to create this value. Module parameter identifies the instance of the module whose executable file contains the icon to be loaded. When you no longer need the icon, call it's Destory destructor to free it.

An application can use the MakeModuleIcon function to access the predefined icons used by the Windows API. To do so, the application must set the Module parameter to zero and the ResID parameter to one of the following values:

IDI_APPLICATION, IDI_ASTERISK, IDI_EXCLAMATION, IDI_HAND, IDI_QUESTION, IDI_WINLOGO (Win95 or NT 4.0 or higher).

If the function fails, the return value is Nil.

MakeModuleIcon example:

```
var
    Ico: TIcon;
    Module: THandle;
begin
    Module := LoadLibrary('MY_RES.DLL');
    if Module >= HINSTANCE_ERROR then
    try
        Ico := MakeModuleIcon(Module, MakeIntResource(2));
        try
            { use icon }
            finally
                 Ico.Free;
                end;
```

finally FreeLibrary(Module); end; end;

Routine MergeForm

Declaration: procedure MergeForm(AControl: TWinControl; AForm: TForm; Align: TAlign; Show: Boolean);

MergeForm merges a form with a NoteBook page.

MergeForm example:

MergeForm(TPage(Notebook.Pages.Objects[0]), TMyForm.Create(Application), alClient, True);

Routine MinimizeText

Declaration: function MinimizeText(const Text: string; Canvas: TCanvas; MaxWidth: Integer): string;

MinimimizeText returns a text string for display purposes, adding ellipses if necessary to maintain a length limit. Call MinimizeText to ensure that a text string fits in the alloted display space. If the text string passed as the Text parameter is longer than MaxWidth pixels when written to the canvas specified by the Canvas parameter, MinimizeText will remove last characters of the string and replace them by ellipses ('...') in order to shorten the resulting string.

MinimizeText example:

{...}
Form1.Canvas.TextRect(Rect, Rect.Left, Rect.Top,
MinimizeText(S, Form1.Canvas, WidthOf(Rect)));
{...}

Routine MsgBox

Declaration: function MsgBox(const Caption, Text: string; Flags: Integer): Integer;

The MsgBox function creates, displays, and operates a message box. This is a wrapper for standard Windows function MessageBox.

MsgBox example:

if MsgBox(Application.Title, 'Continue?', MB_YESNO + MB_ICONSTOP) = mrYes then
begin
{ ... }
end;

Routine NotImplemented

Declaration: procedure NotImplemented;

NotImplemented, as it's name implies, specifies that the code inside a particular procedure implemented is

"not implemented yet."

In other words, the NotImplemented procedure shows a messagebox stating 'Not Implemented Yet'.

NotImplemented example: procedure TForm1.SaveClick(Sender: TObject); begin NotImplemented; end;

Routine PaintInverseRect

Declaration: procedure PaintInverseRect(RectOrg, RectEnd: TPoint);

The PaintInverseRect function inverts a rectangle in a screen device context by performing a logical NOT operation on the color values for each pixel in the rectangle's interior. RectOrg and RectEnd parameters contain the logical coordinates of the rectangle in the screen coordinates (relative to the upper-left corner of the screen).

PaintInverseRect example:

Invert and restore Widows screen: PaintInversRect(Point(0, 0), Point(Screen.Width, Screen.Height)); Delay(2000); PaintInversRect(Point(0, 0), Point(Screen.Width, Screen.Height));

Routine PixelsToDialogUnitsX

Declaration: function PixelsToDialogUnitsX(PixUnits: Word): Word;

Convert pixels to dialog units in horizontal direction. The dialog box base units used by Windows to create dialog boxes.

PixelsToDialogUnitsX example:

X := PixelsToDialogUnitsX(X);

Routine PixelsToDialogUnitsY

Declaration: function PixelsToDialogUnitsY(PixUnits: Word): Word;

Convert pixels to dialog units in vertical direction. The dialog box base units used by Windows to create dialog boxes.

PixelsToDialogUnitsY example:

Y := PixelsToDialogUnitsY(Y);

Routine PointInPolyRgn

Declaration: function PointInPolyRgn(const P: TPoint; const Points: array of TPoint): Boolean;

The PointInRgn function determines whether the specified point is inside the specified region. If the specified point is in the region, the return value is True. Points parameter is an array of TPoint records that define the vertices of the polygon. The polygon is presumed closed. Each vertex can be specified only once.

PointInPolyRgn example:

with MyForm do
if PointInPolyRgn(Point(X, Y), [Point(Left, Top), Point(Left + Width, Top), Point(Left + Width div 2, Top
+ Height div 2)]) then
begin
{ ... }
end;

Routine PointInRect

Declaration: function PointInRect(const P: TPoint; const R: TRect): Boolean;

The PointInRect function determines whether the specified point P lies within the specified rectangle R. A point is within a rectangle if it lies on the left or top side or is within all four sides. A point on the right or bottom side is considered outside the rectangle. If the specified point lies within the rectangle, the return value is True.

PointInRect example:

if PointInRect(Point(X, Y), Image1.ClientRect) then begin
{ ... }
end;

Routine RegisterServer

Declaration: function RegisterServer(const ModuleName: string): Boolean;

RegisterServer registers an OLE server.

RegisterServer example:

RegisterServer('cfx32.ocx');

Routine ResourceNotFound

Declaration: procedure ResourceNotFound(ResID: PChar);

ResourceNotFound triggers an exception 'Resource Not Found' with *ResId* identified (or shown) as the bad resource.

ResourceNotFound example: procedure LoadCursorRes(ResID: PChar; AssignTo: TCursor); var Handle: HCursor; begin Handle := LoadCursor(hInstance, ResID); if Handle = 0 then ResourceNotFound(ResID)

```
else
Screen.Cursors[AssignTo] := Handle;
end;
```

Routine ShadeRect

Declaration: procedure ShadeRect(DC: HDC; const Rect: TRect);

ShadeRect the specified *Rect* (rectangle) area with parameter *DC* (Device to Context).

ShadeRect example: ShadeRect(Canvas.Handle, ARect);

Routine SplitCommandLine

Declaration: procedure SplitCommandLine(const CmdLine: string; var ExeName, Params: string);

SplitCommandLine splits the commandline information presented to the program into the head (ExeName) and tail (Params).

SplitCommandLine example:

S := 'MYAPP.EXE "Parameter Name" -K -S'; SplitCommandLine(S, ExeName, Params); { here ExeName = 'MYAPP.EXE' Params = '"Parameter Name" -K -S' }

Routine StartWait

Declaration: procedure StartWait;

StartWait starts the waiting period. See also: StopWait

StartWait example:

```
StartWait; { set cursor to WaitCursor }
try
{ { ... }
StartWait; { no effect }
try
{ { ... }
finally
StopWait; { no effect }
end;
{ ... }
finally
StopWait; { restore default cursor }
end;
end;
```

Routine StopWait

Declaration: procedure StopWait;

StopWait stops the waiting period. See also: StartWait

StopWait example:

```
StartWait; { set cursor to WaitCursor }
try
{ ... }
StartWait; { no effect }
try
{ ... }
finally
StopWait; { no effect }
end;
{ ... }
finally
StopWait; { restore default cursor }
end;
```

Routine WidthOf

Declaration: function WidthOf(R: TRect): Integer;

Retrieves the horizontal size (width) of the specified rectangle R in pixels.

WidthOf example: W := WidthOf(Rect);

Routine Win32Check

Declaration: function Win32Check(RetVal: Bool): Bool;

Win32Check checks the return value of a Windows 32-bit API call and raises an appropriate exception when it indicates failure.

Call Win32Check with the return value of a Windows API call that returns a Boolean to indicate success or failure. If the Windows API function returns False (indicating failure), Win32Check raise an exception with message returned as

SysErrorMessage(GetLastError).

If the API function returns True, Win32Check returns True.

This function presents only in RX version for Delphi 2.0. In Delphi 3.0 such function presents in standard module SYSUTILS.PAS.

Win32Check example:

Win32Check(CreateDirectory('C:\myworld\private', nil));

UNIT ClipIcon

Const <u>WaitCursor</u>

Routine <u>ActivatePrevInstance</u> <u>ActivateWindow</u> <u>AllocMemo</u> <u>AnsiUpperFirstChar</u>

Type <u>TFillDirection</u> <u>TVertAlignment</u>

Const CF_ICON

Declaration: CF_ICON: Word = 0;;

This is the registered custom clipboard format for Windows icon graphic. This constant can be used to find out if a icon format is available on the Clipboard with the HasFormat method.

Routine AssignClipboardIcon

Declaration: procedure AssignClipboardIcon(Icon: TIcon);

AssignClipboardIcon retrieves icon image from the Clipboard when Clipboard has format CF_ICON, and copies it to a icon object named Icon.

Use the TClipboard.HasFormat method to determine whether the information on the clipboard uses a format compatible with the icon object (CF_ICON). You can place icon in the Clipboard by calling CopyIconToClipboard procedure.

AssignClipboardIcon example:

```
begin
  {...}
  if Clipboard.HasFormat(CF_ICON) then
    AssignClipboardIcon(Application.Icon);
  {...}
end:
```

Routine CopyIconToClipboard

Declaration: procedure CopyIconToClipboard(Icon: TIcon; BackColor: TColor);

CopyIconToClipboard copies the icon to the Clipboard. This function places the data in CF_ICON format and assosiated bitmap to the clipboard. BackColor is color to fill background when creating bitmap from the

icon.

Use CreateIconFromClipboard or AssignClipboardIcon to retrieve icon from the Clipboard.

CopyIconToClipboard example:

var Icon: TIcon; begin ... CopyIconToClipboard(Icon, clBtnFace); end;

Routine CreateIconFromClipboard

Declaration: function CreateIconFromClipboard: TIcon;

CreateIconFromClipboard retrieves icon image from the Clipboard when Clipboard has format CF_ICON, and creates new icon object. You must destroy created object after using it. When Clipboard has not icon data, this function returns nil.

Use the TClipboard.HasFormat method to determine whether the information on the clipboard uses a format compatible with the icon object (CF_ICON).

You can place icon in the Clipboard by calling CopyIconToClipboard procedure.

CreateIconFromClipboard example:

```
var Icon: TIcon;
begin
...
Icon := CreateIconFromClipboard;
try
{ using the icon }
finally
Icon.Free;
end;
...
end;
```

Routine CreateRealSizeIcon

Declaration: function CreateRealSizeIcon(Icon: TIcon): HIcon;

Creates new icon handle from Icon object that has the "real" size. If the function succeeds, the return value is the handle of an icon. If the function fails, the return value is 0.

Before closing, an application must call the Windows API DestroyIcon function to free system resources associated with the icon.

```
CreateRealSizeIcon example:
var
Ico: HIcon;
begin
Ico := CreateRealSizeIcon(Form1.Icon);
try
{ ... }
finally
DestroyIcon(Ico);
end;
end;
```

Routine DrawRealSizeIcon

Declaration: procedure DrawRealSizeIcon(Canvas: TCanvas; Icon: TIcon; X, Y: Integer);

Renders the icon specified by the Icon parameter on the Canvas at the location given by the coordinates (X, Y) using real icon size (for example, you can draw icon 16x16 or 48x48 pixels).

Standard TIcon class always renders icon using Windows metrics SM_CXICON, SM_CYICON.

NOTE. In 16-bit version this procedure is "stub" and is equivalent to standard TCanvas.Draw method.

DrawRealSizeIcon example:

DrawRealSizeIcon(Canvas, Icon, 0, 0);

Routine GetIconSize

Declaration: procedure GetIconSize(Icon: HICON; var W, H: Integer);

The GetIconSize procedure retrieves information about width and height (as W and H parameter correspondingly) of the specified icon.

This procedure work correctly in 32-bit version only. In Delphi 1.0 this procedure is "stub" and always retrieves SM_CXICON, SM_CYICON values.

GetIconSize example: GetIconSize(Icon.Handle, W, H);

Control Name	Unit	Class
GIF Animation Control	<u>GIFCtrl</u>	TRxGIFAnimator

Description:

The TRxGIFAnimator control can display animated GIF images on the form on which it is placed. The animated image is stored in the Image property. It is possible to define number of a frame displayed in the present moment, in runtime on value of property FrameIndex. You can also set this value in the designer. The property Transparent determines, whether the component will be completely transparent (using the TGIFImage.TransparentColor property).

Control also includes properties similar to the TImage's properties. Set Animate property to True to start animation. The OnStart event is generated when the animation started. The OnStop event is generated when the property Active sets to False.

See also:

<u>RX ANI file</u> (TAnimatedCursorImage) and <u>RX Animate</u> (TAnimatedImage), <u>Standard Gif</u> (TRxGif).

Property Animate

Declaration: Animate: Boolean;

Animate indicates whether the animation control is playing the GIF clip. Set Animate to True to cycle through the frames specified by the Image property. Set Animate to False to interrupt the animation control when it is playing the GIF clip. For continuous sequencing of frames, set the Loop property to True.

Property FrameIndex

Declaration: FrameIndex: Integer;

The FrameIndex property stores the frame number of the frame currently displayed. It can be set to the frame number which needs to be displayed. Before using this property, ensure that the GIF image is properly loaded into the Image property. While setting this property, if the value is within the acceptable range (between 0 and the number of frames in the animation, i.e. TGIFImage.Count - 1) then the current frame display is changed and an OnFrameChanged event is generated.

Property Image

Declaration: Image: TGIFImage;

The Image property holds the GIF clip (animated image in GIF89 format).

Property Loop

Declaration: Loop: Boolean;

This property determines whether the GIF animation sequence should be repeated or not when all the frames are displayed. When set to True (default value) and the component runs out of frames, the

FrameIndex property is reset to 0. When Loop property if set to False and the component runs out of frames, the FrameIndex property holds the number of the last displayed frame and the OnStop event is generated.

When the Loop is True use the Animate property to interrupt the animation.

Property Transparent

Declaration: Transparent: Boolean;

Set Transparent to True to replace the background color stored in the GIF file with the parent's background. Set Transparent to False to display the background color stored in the GIF clip.

Event OnFrameChanged

Declaration: OnFrameChanged: TNotifyEvent;

The OnFrameChanged event is generated for every change in the display of frames. The new frame number can be obtained from the FrameIndex property.

Unit <u>GIFCtrl</u> Event <u>OnFrameChanged</u>

Property Animate FrameIndex Image Loop Transparent

Control Name	Unit	Class
GIF	RxGIF	TGIFFrame

Description:

TGIFImage supports reading and writing GIF compressed image data. The GIFImage object uses the data from an instance of a TGIFData object, which contains that actual GIF data source and is never modified. Each GIF image object may share this data source object with other instances of a GIF image that are copies of it, created by using the Assign method. The GIF data source does referencing counting for the GIF image objects that are linked to it

The properties of TGIFImage indicate the size and characteristics of the GIF image.

TGIFImage supports up to 256-colors GIF images versions 87a and 89a (you can determine version of loaded image by Version property), and supports graphic control extensions (multiple images, transparency) and comment extensions of GIF format. Plain text extensions are not supported.

Single image from GIF animation sequence can be accessible by the Frames property.

TGIFImage has an internal bitmaps for each frame that contains the GIF image. These internal images and the original data source of a GIF image are read only.

A TGIFImage object:

• Has no canvas (and thus it cannot draw onto a canvas). A TGIFImage object does implement the protected Draw method introduced in TGraphic. This means that it can draw itself. When TGIFImage contains multiple frames (animated GIF) the Draw method will be draw frame specified by FrameIndex property.

• Provides no access to the internal bitmap image that it creates for the GIF image.

• TGIFImage and TGIFFrame can be assign to TBitmap object and you can assign TBitmap object to the TGIFImage or TGIFFrame. So you can create TGIFImage, assign bitmap to it and then store image as GIF file by using SaveToFile method.

Property BackgroundColor

Declaration: BackgroundColor: TColor;

The BackgroundColor property determines which background color stored in the GIF file to use when drawing a GIF image. The background color is the color used for those pixels on the screen that are not covered by an image.

Property Comment

Declaration: Comment: TStrings;

This contains the comment text stored in comment extension of a GIF file.

Property Count

Declaration: Count: Integer;

This is a read-only property and for animated GIFs. It stores the number of frames contained within the GIF file.

Property FrameIndex

Declaration: FrameIndex: Integer;

This property contains the current frame being displayed. When this property is changed, it would change the frame being displayed on the canvas.

Property Frames[Index: Integer]

Declaration: Frames[Index: Integer]: TGIFFrame;

Procedure returns the GIF frame from the index. This is used to obtain (or extract) a particular frame from the GIF animated file.

Property Height

Declaration: Height: Integer;

Height specifies the vertical size in pixels of the current frame (specified by FrameIndex property) of the GIF image.

Property Palette

Declaration: Palette: HPALETTE;

Palette indicates the color palette of the graphical GIF image. Use Palette to get the color palette of a GIF image.

Property ScreenHeight

Declaration: ScreenHeight: Integer;

Logical screen height, in pixels, of the Logical Screen where the images will be rendered in the displaying device. Specifies the height of the space in which the animation plays. This value is a taller space for frames that move vertically.

Property ScreenWidth

Declaration: ScreenWidth: Integer;

Logical screen width, in pixels, of the Logical Screen where the images will be rendered in the displaying device. Specifies the width of the space in which the animation plays. This value is a wider space for frames

that move horizontally.

Property TransparentColor

Declaration: TransparentColor: TColor;

TransparentColor determines which color of the GIF image is to be transparent when the GIF image is drawn.

Property Version

Declaration: Version: TGIFVersion;

Version number used to format the GIF data stream. Identifies the minimum set of capabilities necessary to a decoder to fully process the contents of the Data Stream. GIF version numbers can be one of following values:

- gv87a May 1987
- gv89a July 1989.

Property Width

Declaration: Width: Integer;

Width specifies the horizontal size in pixels of the current frame (specified by FrameIndex property) of the GIF image.

Event OnProgress

Declaration: OnProgress: TProgressEvent;

OnProgress occurs when a graphical image is in the process of changing. OnProgress is a generic progress indicator event that propagates out to the TGIFImage OnProgress events.

Type TDisposalMethod

Declaration: TDisposalMethod = (dmUndefined, dmLeave, dmRestoreBackground, dmRestorePrevious, dmReserved4, dmReserved5, dmReserved6, dmReserved7);

TDisposalMethod is the type of available values for the DisposalMethod property of the TGIFFrame object.

Type TGIFVersion

Declaration: TGIFVersion = (gvUnknown, gv87a, gv89a);

TGIFVersion defines the types of GIF file that can be read.

Property AnimateInterval

Declaration: AnimateInterval: Word;

Specifies the amount of time, in milliseconds, that the image displays during the animation.

Property Bitmap

Declaration: Bitmap: TBitmap;

An internal bitmap that contains the GIF frame's image. Protected.

Property DisposalMethod

Declaration: DisposalMethod: TDisposalMethod;

DisposalMethod indicates how frames display in the animation from the following choices:

• dmUndefined - Directs the viewer (animator) to do nothing to the background before displaying the next image.

• dmLeave - Directs the viewer to leave the previous graphic image as the next is drawn.

This choice can create a shadowing effect.

• dmRestoreBackground - Directs the viewer to redraw the original background as the current image is drawn.

• dmRestorePrevious - Directs the viewer to redraw the previous image as the current image is drawn

Property Height

Declaration: Height: Integer;

Height indicates the vertical size in pixels of the GIF-frame image.

Property Origin

Declaration: Origin: TPoint;

Origin indicates the position of the left and top edges of the image, with respect to the left edge of the Logical Screen. Leftmost column of the Logical Screen is 0. Top row of the Logical Screen is also 0.

Property TransparentColor

Declaration: TransparentColor: TColor;

TransparentColor is a color that will treat as the transparent portion of the GIF image.

Property Width

Declaration: Width: Integer;

Width indicates the horizontal size in pixels of the GIF-frame image.

Method Draw

-

Declaration: procedure Draw(ACanvas: TCanvas; X, Y: Integer);

Draw the GIF frame on the canvas specified by the ACanvas parameter at the location given by the coordinates (X, Y).

Unit **<u>RxGIF</u> Property: AnimateInterval BackgroundColor Bitmap Comment** Count **DisposalMethod** FrameIndex Frames[Index: Integer] **Height Origin Palette** ScreenWidth **TransparentColor Version Width**

Types: <u>TDisposalMethod</u> <u>TGIFVersion</u>

Events: OnProgress

Method: Draw

Control Name	Unit	Class
RxDbCalcEdit	RxDBCtrl	TRxDBCalcEdit

Description: The TRxDBCalcEdit component is the data-aware version of the TRxCalcEdit. This means that the TRxDBCalcEdit can be connected to a data source and edit the contents of numeric fields in the data source. TRxDBCalcEdit provides a drop-down calculator to help you calculate the number. TRxDBCalcEdit is a direct descendant of the TRxCalcEdit class and inherits all of its properties, events and methods.

Property Value

Declaration: Value: Extended;

Value is the actual data in a TRxDBCalcEdit. Use Value to read data directly from and write data directly to an editor. When Text property is empty string, the Value property returns 0 (zero).

Control Name	Unit	Class
RX Calculator	<u>RxCombos</u>	TRxCalcEdit

Description:

The TRxCalcEdit takes the display and editing of numeric data one step further: it provides a popup calculator to help you calculate the number. When you perform a calculation on the calculator, the result is inserted into the number edit's display.

TRxCalcEdit have the keyboard restrictions, so the user cannot enter invalid characters. When the user leaves the field, the number is reformatted to display appropriately, accordingly with the DisplayFormat property.

The field value is stored in a write <u>Value</u> property so you should read and write to that in your program.

This field is of type Extended. You can also read and write <u>Value</u> as integer number using AsInteger property.

Property GlyphKind

Declaration: GlyphKind: TGlyphKind;

Use this property to select between standard button glyph types, or specify a custom type.

The standard types are:

- gkCustom the button is displayed with a custom glyph, specified by Glyph property;
- gkDefault the button is displayed with a glyph, dependent on the editor class (TFilenameEdit, TDateEdit etc);
- gkDropDown the button is displayed with the standard combo box drop down glyph;
- gkEllipsis the button is displayed with an ellipsis glyph.

If the Glyph is set to a new bitmap, the gkCustom type is automatically set.

Property Value

Declaration: Value: Extended;

Value is the actual data in a TCurrencyEdit or in a TRxCalcEdit. Use Value to read data directly from and write data directly to an editor. When Text property is empty string, the Value property returns 0 (zero)

UNIT RxGraph

Overview: <u>BitmapToMemory</u> <u>TMappingMethod</u> <u>GetBitmapPixelFormat</u> <u>GrayscaleBitmap</u> <u>SaveBitmapToFile</u> <u>SetBitmapPixelFormat</u>

Type TMappingMethod

Declaration: TMappingMethod = (mmQuantize, mmTrunc784, mmTrunc666, mmTripel, mmGrayscale);

The TMappingMethod describes possible methods to decrease bitmap color count by using SetBitmapPixelFormat procedure.

Routine BitmapToMemory

Declaration: function BitmapToMemory(Bitmap: TBitmap; Colors: Integer): TStream;

BitMap : pointer to bitmap Colours: Number of colours. (maximum: 2^32)

This maps a bitmap to a stream to be utilitsed by a memory mapped function or for file streaming purposes.

BitmapToMemory example:

Next code fragment shows how you can decrease the number of colours in a bitmap to 16 colours:

```
var
S: TStream;
begin
S := BitmapToMemory(Picture.Bitmap, 16);
try
Picture.Graphic.LoadFromStream(S);
finally
S.Free;
end;
end;
```

Routine GetBitmapPixelFormat

Declaration: function GetBitmapPixelFormat(ABitmap: TBitmap): TPixelFormat;

The GetBitmapPixelFormat procedure determines in what bit format the bitmap image is displayed.

```
> GetBitmapPixelFormat example:
```

if GetBitmapPixelFormat(Image.Bitmap) > pf8bit then SetBitmapPixelFormat(Image.Bitmap, pf8bit, mmQuantize);

Routine GrayscaleBitmap

Declaration: procedure GrayscaleBitmap(Bitmap: TBitmap);

This procedure transforms a color bitmap image into grayscale. Each pixel is transformed into the gray pixel (Red = Green = Blue) with the (approximate) same luminosity.

GrayscaleBitmap example: GrayscaleBitmap(Image1.Bitmap);

Routine SaveBitmapToFile

Declaration: procedure SaveBitmapToFile(const FileName: string; Bitmap: TBitmap; Colors: Integer);

SaveBitmapToFile example:

var Bmp: TBitmap; begin {...} Bmp.Assign(Clipboard); SaveBitmapToFile(SaveDialog1.FileName, Bmp, 16); {...} end;

Routine SetBitmapPixelFormat

Declaration: procedure SetBitmapPixelFormat(ABitmap: TBitmap; PixelFormat: TPixelFormat; Method: TMappingMethod);

SetBitmapPixelFormat procedure allows you to determine in what bit format the bitmap image specified by ABitmap parameter will be displayed: 8-bit, 24-bit etc. For example, you can use SetBitmapPixelFormat to set the pixel format of the bitmap image to 8-bit for video driveres that cannot display the native format of a bitmap image. You can specify method to decrease bitmap colors count by the Method parameter. Possible values for PixelFormat parameter are: pf1bit, pf4bit, pf8bit, pf24bit.

SetBitmapPixelFormat example:

SetBitmapPixelFormat(Image.Bitmap, pf8bit, mmQuantize);

Copyright © 1992-2000 by Peck Kim Han <u>http://www.hypermart.net</u>. All rights reserved. The software which is documented in this help file is copyrighted by their respective owners.

VCL Utilities