# About Win32++

This documentation describes the features of Win32++, version 7.8. To download a copy of Win32++ proceed to the project's hosting at SourceForge <u>here</u>.

Win32++ is a C++ library used to build windows applications. Win32++ is a free alternative to MFC. It has the added advantage of being able to run on a wide range of free compilers, including Visual Studio Express, and the MinGW compiler provided with CodeBlocks and Dev-C++.

Win32++ provides a thin layer over the Windows API and is similar to MFC. There are several differences to MFC however. Win32++ uses very few macros, and doesn't use macros for message maps. All the code for Win32++ is provided in a set of header files. There is no need for any additional DLLs or libraries apart from those DLLs that are part of the window operating system. Win32++ has view windows but doesn't impose a Doc/View architecture on users. User are free to employ a a Doc/View architecture if they choose.

Win32++ supports all Windows operating systems, from Windows 95 through to Windows 10 and Windows Server 2012. It also supports the Windows CE operating system which runs on runs on the various Pocket PCs, Smartphones, as well as industrial devices and embedded systems.

This documentation is intended to be used along side the documentation for the Windows Application User Interface (API). The Windows API documentation ships as part of the Microsoft Platform SDK and also the Microsoft Windows SDK. The Platform SDK is targeted at Windows XP and earlier. The Windows API is targeted at the more modern versions of Windows.

<u>Microsoft Platform SDK</u> <u>Microsoft Windows SDK</u>

# Win32++ Overview

# Introduction

Win32++ consists of a C++ library used to create window applications. It is a free alternative to MFC, and can be used on a larger range of compilers including those from Borland, Microsoft and MinGW. The library can be used to create a variety of programs, including simple windows, dialogs, property sheets, as well as frame and MDI frame applications. Information on how to use these classes to build your own Win32 applications is available in the <u>Tutorial</u> section.

Frame based applications usually have an outer frame window which looks after the menu, toolbar and status bar, and a separate window (sometimes referred to as a view window) over the client area. Many of the windows applications you are familiar with are likely to be frame based applications. You can use this library to produce not just simple windows and dialogs, but also professional looking frame based applications, complete with a menu, toolbar, status bar, and tooltips.

# About the Downloads

To download a copy of Win32++ proceed to the <u>Download</u> section of the projects listing at SourceForge. The <u>Downloads</u> section provides a number of sample applications which use Win32++, including:

- Browser
- CustomControl
- DateTime
- Dialog
- DialogBars
- DialogDemo
- DialogResizing
- DialogTab
- DirectX
- DLL
- Dock
- DockContainer
- DockTabbedMDI
- Explorer
- Fast GDI
- FormDocView
- Frame
- GDIPlus
- MDI Frame
- MDI Frame Demo
- MDI Frame Splitter
- Networking

- Notepad
- Performance
- Picture
- PropertySheet
- RebarDemo
- RibbonFrame
- RibbonSimple
- Scribble
- Simple Window
- Splitter
- StatusBar
- Tab Demo
- TaskDialog
- Themes
- Threads Demo
- ToolBar Demo
- Tray Demo

# Features

Win32++ provides a library for developing applications, using the Windows API directly. It supports all MS operating systems which run 32bit and 64bit Windows, from Windows 95 through to Windows 10 and Server 2012. Win32++ is designed to produce programs with a similar look and feel to those created using MFC. This library can develop applications based on simple windows, dialogs, frames and MDI frames. The frames produced by Win32++ have the following features:

- Rebar control
- MenuBar (a menu housed inside the rebar control).
- ToolBar
- Status bar
- Tool tips
- Docking

Win32++ also brings an object oriented approach to programming directly with the Windows API. Each window created is a C++ class object capable of having its own window procedure for routing messages.

A summary of the features of Win32++ are as follows:

- Support for both Win32 and Windows CE operating systems. It supports all Win32 operating systems, from Windows 95 through to Windows 10. The Windows CE operating systems supported range from WCE400 (Windows mobile 2003) to WCE600 (Windows mobile 6).
- Supports a wide range of C++ compilers, including those from Borland and Microsoft, as well as the GNU compiler used in Dev-C++.
- Supports applications based on simple windows, dialogs, forms, property sheets, SDI frames and MDI frames. The SDI and MDI frames include rebars, status bars, menu bars, toolbars and tooltips. The frames use a separate window, often referred to as a view window, for the frame's client area.
- Support for docking, tabbed containers and TabbedMDIs.
- Message routing. The messages for each window are routed to the WndProc function of the associated CWnd object. All messages are passed to the WndProc function, including all those associated with window creation.
- Automatic Subclassing. When creating a window using a predefined class type, Win32++ automatically subclasses the window so that messages can be routed via WndProc. Predefined class

types include all the common controls, such as the toolbar, status bar, tree-view etc.

- Existing windows can be attached to CWnd objects. This is particularly useful for controls in dialogs. These can be attached to to CWnd objects with the AttachDlgItem function.
- Notification reflection. When a notification (WM\_NOTIFY message) is received from a child window, OnNotifyReflect is called for the CWnd object associated with the child window.
- Message reflection. Older common controls pass messages other than WM\_NOTIFY to notify the parent window of events. The OnMessageReflect is called to pass these messages back to the CWnd object associated with the child window.
- Network support. Win32++ provides a CSocket class which is a thin wrapper for much of the Winsock SPI. This class also provides notification of network events. Supports IPv4 and IPv6.
- Tracing. Trace output is sent to the Output pane of the IDE when the program is compiled with debugging enabled.
- Multi-Thread support. Win32++ is thread safe, and windows can be created in separate threads.
- 64 bit support. Win32++ can be used to create 64bit applications.
- Multilingual support. SetResourceHandle can be used to load resources from a resource dll to provide support for different languages.
- Support for Task Dialogs available on the Windows Vista and later operating systems.
- Support for the Ribbon Framework available on Windows 7.

# **Requirements for using the Win32++**

In order to use Win32++, you will need the following:

- A C++ compiler.
- Some knowledge of C++, including how to use classes
- Some knowledge of the Windows API

You should also have a copy of the Microsoft Platform Software Development Kit, as this includes the documentation for programming with the Windows API. You will find this, as well as a selection of free C+ compilers and Windows API tutorials on the <u>links</u> page.

# **Getting Started**

The library itself can be downloaded <u>here</u>. Simply start with one of the samples provided. The samples include the project files for Visual Studio 6, Visual Studio.Net 2003, VS2005 Express, VS2008 Express, VS2010 Express, VS2013 Express, Dev-C++ and CodeBlocks. If you're using one of these, the samples are pre-configured, and ready to compile.

If you're new to C++ programming and looking for a free compiler to use, I recommend Microsoft's Visual C++ Express. I found this compiler to be the easiest free compiler to set up and use. If you are using an older Microsoft's Visual Studio 6.0 compiler you may also need to install and incorporate the Platform SDK. Information on how to do this is provided by Microsoft <u>here</u>. To compile the DirectX sample you will need to install the DirectX SDK.

Using Win32++

This section describes how to use Win32++ to build window applications.

- <u>Getting Started</u>
- <u>Simple Windows</u>
- <u>Dialogs</u>
- <u>Frames</u>
- MDI Frames
- Graphics Device Interface (GDI)
- <u>Docking</u>
- <u>Text Conversion Functions</u>
- <u>CString</u>

# **Getting Started**

### Before we begin

In order to use Win32++ you will need a C++ compiler. If you don't already have one, there are several free C++ compilers available for download from the internet. Some of the more popular choices are:

- <u>Microsoft's Visual C++ Express</u>
- <u>Code::Blocks</u>
- <u>Dev-C++</u>

Different compilers have their various strengths and weaknesses. My personal favourite is Microsoft's Visual C++ 2008 Express. If you use Visual Studio 2010 Express, be sure to enable the "Expert Settings" to access features such as the Class View, and the ability to clean and rebuild a project.

You should also download and install a copy of the Windows Software Development Kit (SDK). The Windows SDK includes the documentation of the Windows API, which is the primary reference document for windows programming. This is a free download available from Microsoft. The Windows SDK can be downloaded at: <u>http://msdn.microsoft.com/en-us/windows/bb980924.aspx</u>

Finally if you are planning to develop dialog applications or work with resources, you will need a resource editor. The commercial compilers generally include a resource a resource editor, but the free ones don't. Fortunately there are some free resource editors available. One such resource editor is ResEdit, available for download at: <a href="http://www.resedit.net/">http://www.resedit.net/</a>

#### Installing Win32++

Win32++ is available for download as a zip archive from SourceForge at <u>http://sourceforge.net/projects/win32-framework/</u>. Extract the files into a directory of your choosing, perhaps a subdirectory of your documents folder. When extracting the files, be sure to retain Win32++'s directory structure.

The Win32++ zip archive contains the Win32++ library files in the "include" directory, as well as a number of program samples which demonstrate the various type of applications that Win32++ can be used to create. Each sample contains a "ProjectFiles" directory, which contains a collection of project files matching a range of commonly used compilers. To compile the sample code, run the project which matches your compiler.

#### Creating a new project

Perhaps the simplest way to create a new project is to take one of the samples, and copy it to a new directory within the same parent directory as the samples. Alternatively you could use one of the project files from the "Win32++\new projects" directory which are provided for this purpose.

If you are looking to create your own project from scratch, you will typically need to do the following:

- Start with a Win32 GUI application
- Add "..\..\include" to the C++ Additional Include Directories.

- Add "..\..\include" to the Resources Additional Include Directories.
- Add additional libraries to the Linker Additional Dependencies (usually comctl32.lib).

You can use the existing project files as a guide as to how this should be done.

#### **A Simple Window Application**

The following code is a complete program for creating and displaying a window using Win32++. This program creates a simple window, and ends the application when the window is closed.

```
// main.cpp
// Add the Win32++\include directory to project's additional include di
#include "wincore.h"
class CView : public CWnd
{
public:
   CView() {}
   virtual ~CView() {}
   virtual void OnDestroy() { PostQuitMessage(0); } // Ends the appl
};
int APIENTRY WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
   // Start Win32++
   CWinApp MyApp;
   // Create our view window
   CView m View;
   m View.Create();
   // Run the application
   return MyApp.Run();
}
```

In this code we inherit the CView class from CWnd to represent our view window. CView overrides the OnDestroy function which is called when the window is destroyed. There we call PostQuitMessage which ends the application.

#### **The Tutorials**

The tutorials contain a step by step guide to building a windows application using Win32++. The description of the each tutorial is contained within this help document, and the code is provided in the "tutorials" folder.

# **Simple Window**

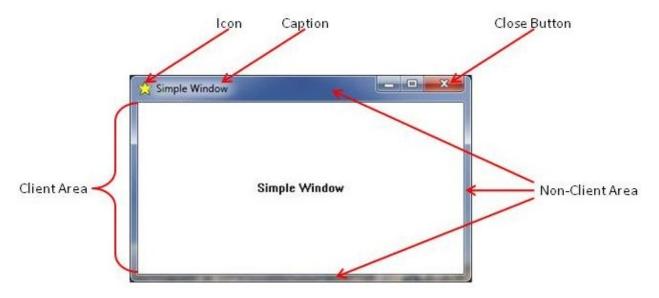
### Description

A window typically occupies a rectangular portion of the screen. It is managed by the Windows operating system, and is capable of responding to events and redrawing itself. A window has a handle (often referred to as a HWND) through which a program can control the window or send it messages.

There are many different types of windows. In this section we will focus on a simple application window. The following sections will focus on more specialized widows, such as dialogs, toolbars, status bars, edit controls and so forth.

In Win32++, the CWnd class is used to create and control windows. We inherit a class from CWnd and override OnPaint to control the way our window looks, and override WndProc to control how it responds to messages. Refer to the description of CWnd in the Class Library section of this documentation for a description of the functions which can be overridden.

#### Components



The following diagram illustrates the components of a typical application window.

Typically an application will draw over the client area of the window, but leave the drawing of the nonclient area to the operating system. Note that a child window (a window contained within another window) might not have a visible non-client area.

This diagram was created from an image capture of the window created by the "Simple" sample (one of the samples that ship with Win32++).

#### **Creating a Window**

Its important to note that the CWnd's window isn't created when the CWnd is constructed. The window is created some time later, when either the Create or the CreateEx function is called.

Normally the Create function is used to create the window. This function allows us to specify the parent window, if any. It also calls PreRegisterClass to set the window class parameters, and PreCreate to set the creation parameters before creating the window. The PreRegisterClass and PreCreate can be overridden to set these parameters prior to window creation, as shown below.

```
void CView::PreRegisterClass(WNDCLASS &wc)
{
    // Set the background brush, class name and cursor
    wc.hbrBackground = m_hBrush;
    wc.lpszClassName = _T("Scribble Window");
    wc.hCursor = ::LoadCursor(GetApp()->GetResourceHandle(), MAKEINTRESOUR
}
```

Additional notes for PreRegisterClass:

- The lpszClassName must be set for this function to take effect.
- The lpfnWndProc is always CWnd::StaticWindowProc.
- The styles that can be set here are "class" styles. These are a different set of styles to those set by CREATESTRUCT (used in PreCreate).
- Default class creation parameters are not set when PreRegisterClass is used, so the following settings might prove useful:
  - wc.hCursor = ::LoadCursor(NULL, IDC\_ARROW);
  - wc.hbrBackground = (HBRUSH)::GetStockObject(WHITE\_BRUSH);
  - o wc.hIcon = ::LoadIcon(NULL, IDI\_APPLICATION);

```
void CView::PreCreate(CREATESTRUCT &cs)
{
    // Set the extra style to provide a sunken effect
    cs.dwExStyle = WS_EX_CLIENTEDGE;
}
```

Additional notes for PreCreate:

- The cs.lpszClass defaults to the class name specified in PreRegisterClass, or "Win32++ Window" otherwise.
- The styles that can be set here are "window" styles. These are a different to the set of "class" styles used in PreRegisterClass.
- A default style of WS\_VISIBLE and possibly WS\_OVERLAPPEDWINDOW is used if no style is specified. Be sure to add these styles if required when specifying the window style.

Create is the function which would normally be used to create a window, but sometimes it is more convenient to take more direct control over the various window creation parameters. CreateEx is the function which allows the various window parameters to be set directly when the window is created.

The OnCreate function is called during window creation. This function can be overridden to perform addition tasks during window creation, such as creating child windows. The OnInitialUpdate function is called once window creation is complete, so this function can be overridden to perform tasks after the window has been successfully created.

#### **Painting the Window**

In order to paint on the window, we override OnDraw and put our drawing code there. OnDraw is called automatically whenever all, or a part of the window needs to be redrawn. OnDraw provides us with a pointer to the client area's device context. All normal drawing for a window is done via a device context.

The following code loads a text string from a string resource and centers it on the client area of the window.

```
void CView::OnDraw(CDC* pDC)
{
    // OnDraw is called automatically whenever a part of the window needs
    // Centre some text in our view window
    CRect rc = GetClientRect();
    pDC->DrawText(LoadString(IDW_MAIN), rc, DT_CENTER|DT_VCENTER|DT_SINGLE
}
```

### **Handling Messages**

Each CWnd handles the messages for its window. Override WndProc to specify how the CWnd should handle messages. Any unhandled messages are passed on to the default window procedure.

A WndProc function could look something like this:

```
LRESULT CView::WndProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
{
 // This function is our message procedure. We process the messages for
 // the view window here. Unprocessed messages are passed on for
 // default processing.
 switch(uMsg)
  {
 case WM DESTROY:
    OnDestroy();
    return 0; // return a value. No default processing
 case WM SIZE:
    OnSize();
                // and also do default processing for this message
    break:
 }
 // pass unhandled messages on for default processing
  return WndProcDefault(uMsg, wParam, lParam);
}
```

# **Dialogs**

### Description

A dialog or dialog box is a specialised window, designed to host other controls, such as buttons, toolbars, tree-views, list-views and so forth. A dialog is typically used to gather information from or present information to the user.

In Win32++, the <u>CDialog</u> class is used to create and control dialogs. We inherit a class from <u>CDialog</u> and override functions to control the way the dialog interacts with its controls. Refer to the description of <u>CDialog</u> in the Class Library section of this documentation for a description of the functions which can be overridden.

#### **Dialog Components**

The following picture shows a dialog with several controls, namely: a slider control; a scrollbar control; a progress bar control, a static control (for text); and a button control. These controls are actually specialised windows, and they are child windows of the dialog.

Scroll bar Control	Slider Control	Static Control Scroll Position 38
Progress bar Control		
	Progress bar Control	

#### **Modal and Modeless Dialogs**

The behaviour of dialog boxes vary, depending on whether the dialog box is modal or modeless. A modal dialog box requires the user to close the dialog box before activating another window in the application. A modeless dialog box does not require an immediate response from the user. It is similar to a main window containing controls.

#### Working with resources

The resources used in an application are defined in a resource script. In the Win32++ samples this file is

called Resource.rc. The resource script can be used to define the following types of resources.

- Accelerator
- Bitmap
- Cursor
- Dialog
- Icon
- Manifest
- Menu
- String table
- User defined
- Version Information

Typically a resource editor is used to create the resource script. The commercial compilers generally include a resource a resource editor, but the free ones don't. Fortunately there are some free resource editors available. One such resource editor is ResEdit, available for download at: <u>http://www.resedit.net/</u>.

When resources are defined, they will have an associated resource ID. These resource IDs are defined in a header file (typically resource.h). The resource script and its associated header file should be added to your project. When building your project, the resource script is compiled and linked to your executable or dll.

Its important to note that the resources defined in the resource script are actually embedded in your program. A very large resource, (a large bitmap for example) can increase the size of the executable considerably. If the size of the executable is an important consideration, large bitmaps and icons could be loaded from files during runtime rather than defined in the resource script.

### **Creating the dialog**

The first step in creating the dialog is to define the dialog resource using a resource editor. The resource ID defined in this process is then used to construct the object inherited from <u>CDialog</u>.

This is an example of how a dialog would be constructed for a simple dialog application. The CDialogApp is inherited from <u>CWinApp</u>, and would be instanciated in WinMain. CDialogApp's constructor initialization list provides the resource ID of the dialog to CMyDialog's constructor. When the application runs, InitInstance is called which creates a modal dialog.

```
// Declaration of the CDialogApp class
class CDialogApp : public CWinApp
{
public:
    CDialogApp() : m_MyDialog(IDD_DIALOG1) {}
    virtual ~CDialogApp() {}
    virtual BOOL InitInstance();
    {
        //Display the Modal Dialog
        m_MyDialog.DoModal();
        //End the program
        ::PostQuitMessage(0);
        return TRUE;
    }
```

```
CMyDialog& GetDialog() {return m_MyDialog;}
private:
CMyDialog m_MyDialog;
};
```

#### Working with controls

The OnInitDialog function is called when the dialog is created. We can override this function to determine what happens when the dialog starts. In this example the dialog's icon is set, and one of the dialog's controls (a button) is attached to a class inherited from <u>CButton</u>.

```
B00L CMyDialog::OnInitDialog()
{
    // Set the Icon
    SetIconLarge(IDW_MAIN);
    SetIconSmall(IDW_MAIN);
    // Attach the dialog controls
    AttachItem(IDC_BUTTON1, m_Button);
    return true;
}
```

Attaching the button control to m\_Button like this allows us to intercept the messages for the button, and handle them in m\_Button's WndProc function.

```
LRESULT CMyButton::WndProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
{
  switch (uMsg)
  {
    case WM MOUSEMOVE:
                             OnMouseMove(wParam, lParam);
                                                                  break;
    case WM NCHITTEST:
                             OnNCHitTest(wParam, lParam);
                                                                  break;
    case WM SETCURSOR:
                             OnSetCursor(wParam, lParam);
                                                                  break;
    case WM LBUTTONDOWN:
                             OnLButtonDown(wParam, lParam);
                                                                  break;
    case WM LBUTTONUP:
                             OnLButtonUp(wParam, lParam);
                                                                  break;
    case WM RBUTTONDOWN:
                             OnRButtonDown(wParam, lParam);
                                                                  break;
    default:
      TRACE("CMyButton::WndProc - Unspecified Message\n");
      break;
  }
  // Pass all other messages on for default processing
  return WndProcDefault(uMsg, wParam, lParam);
}
```

**Responding to Events** 

When a button is pressed on a dialog, a WM\_COMMAND message is sent to the dialog window. Override OnCommand to respond to these commands as demonstrated in the following code.

```
BOOL CMyDialog::OnCommand(WPARAM wParam, LPARAM /*lParam*/)
Ł
  switch (LOWORD(wParam))
  {
    case IDC BUTTON1:
                                          return TRUE;
                          OnButton();
    case IDC RADI01:
                          OnRadio1():
                                          return TRUE:
    case IDC RADIO2:
                          OnRadio2();
                                          return TRUE;
                          OnRadio3();
OnCheck1();
    case IDC RADI03:
                                          return TRUE;
    case IDC CHECK1:
                                          return TRUE;
    case IDC CHECK2:
                          OnCheck2();
                                          return TRUE;
    case IDC CHECK3:
                          OnCheck3();
                                          return TRUE;
  }
  return FALSE;
}
```

The Ok and Cancel buttons call the OnOK and OnCancel functions when pressed. The default behaviour of these functions is to close the dialog. Override these functions to perform other tasks.

### **Handling Notifications**

Common controls (for example an Edit control) send notification messages to their parent window to notify them of events, usually by way of a WM\_NOTIFY message. The CDialog's OnNotify function is called in response to a WM\_NOTIFY message. It can be overridden to handle these notification events.

OnNotify is used to handle notification messages in the class inherited from CDialog. However, we can use the OnNotifyReflect to handle the notification in the <u>CWnd</u> of the child window that generated the notification instead. It is a matter of personal preference whether to use OnNotify to handle the notification in the parent (dialog) or OnNotifyReflect to handle it in the child control that created it. If we are handling notifications from several controls, OnNotifyReflect can prove simpler.

When overriding the OnNotifyReflect function, it could look something like this:

```
LRESULT CMyTreeView::OnNotifyReflect(WPARAM, LPARAM lParam)
{
  LPNMHDR lpnmh = (LPNMHDR) lParam;
  switch(lpnmh->code)
  {
   case NM_RCLICK:
    // Handle the mouse right button click notification
    {
      CPoint ptScreen = GetCursorPos();
      DoContextMenu(ptScreen);
    }
    break;
} //switch LPNMHDR
```

```
return 0L;
```

}

Likewise, OnMessage and OnMessageReflect can also be used to handle a small set of older messages that behave like notifications. The set of messages which are handled by OnMessage and OnMessageReflect are as follows:

WM\_CTLCOLORBTN WM\_CTLCOLOREDIT WM\_CTLCOLORDLG WM\_CTLCOLORLISTBOX WM\_CTLCOLORSCROLLBAR WM\_CTLCOLORSTATIC WM\_DRAWITEM WM\_DRAWITEM WM\_MEASUREITEM WM\_DELETEITEM WM\_DELETEITEM WM\_CHARTOITEM WM\_CHARTOITEM WM\_VKEYTOITEM WM\_HSCROLL WM\_VSCROLL WM\_PARENTNOTIFY

When overriding OnMessageReflect, it might look something like this:

```
LRESULT CMySlider::OnMessageReflect(UINT uMsg, WPARAM wParam, LPARAM lPa
{
  UNREFERENCED PARAMETER(wParam);
  UNREFERENCED PARAMETER(lParam);
  switch (uMsg)
  {
  case WM HSCROLL:
    {
      // Get the slider bar position
      int nPos = GetPos();
      // Get a pointer to the MyDialog object
      CMyDialog* pDialog = (CMyDialog*)GetParent();
      pDialog->SetProgress(nPos);
                                                 // Set the progress bar
      pDialog->SetScroll(nPos);
                                        // Set the scroll bar position
```

```
pDialog->SetStatic(TRUE, nPos); // Set the static text
            break;
        }
        return 0;
}
```

## Handling the Dialog's Messages

The dialog itself is a window. It is the parent window for each of the controls. Sometimes we need to handle the dialog's messages too. When we need to handle the dialog window's messages, we override DialogProc to specify how the messages should be handled. Any unhandled messages are passed on to the default dialog procedure.

When overriding the DialogProc function, it could look something like this:

```
B00L CMyDialog::DialogProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    // switch (uMsg)
    // {
        // //Additional messages to be handled go here
        // //Additional messages on to parent DialogProc
        return DialogProcDefault(uMsg, wParam, lParam);
}
```

# Frames

## Description

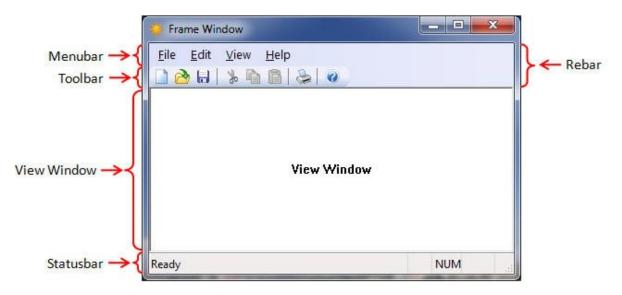
Frames provides application developers with a standard way of presenting their applications to users. Frames usually display a menu, and often also display a toolbar and status bar.

There are two styles of frames used in Microsoft Windows. The Single Document Interface (SDI) frame allows the application to display a single view window. The Multiple Document Interface (MDI) frame allows the application to manage and display multiple view windows at the same time. The frames described in this section are SDI frames.

Refer to <u>CFrame</u> in the Class library section for a description of its member functions.

#### **Frame components**

The following diagram illustrates a typical frame application.



The toolbar and menubar are used to accept input from the user. The status bar is used to display status information. The view window is a child window of the frame, positioned over the area that is not occupied by the menu, toolbar and status bar.

#### **Defining the Frame**

To create a CMainFrame class, override Win32++'s CFrame class as follows:

```
// Declaration of the CMainFrame class
class CMainFrame : public CFrame
{
public:
```

```
CMainFrame(void);
virtual ~CMainFrame();
protected:
virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
virtual void OnCreate();
virtual void OnInitialUpdate();
virtual void OnInitialUpdate();
virtual LRESULT OnNotify(WPARAM wParam, LPARAM lParam);
virtual void SetupToolBar();
virtual LRESULT WndProc(UINT uMsg, WPARAM wParam, LPARAM lParam);
private:
CView m_View; // The view window
};
```

#### The View Window

Every frame must be provided with a view window. The view window can be any sort of window, providing it is a child window, and it is capable of being resized. The types of windows that might be used as view windows include simple windows, rich edit controls, tree view controls, list view controls, tab controls, modeless dialogs, Win32++ dockers, etc.

The view is a window class inherited from <u>CWnd</u>. The view window must be set before the frame window is created. This is usually done in the constructor as follows:

```
CMainFrame()
{
   //Set m_View as the view window of the frame
   SetView(m_View);
}
```

Incidentally, other parts of Win32++ use view windows as well. View windows are used in:

- Frames
- MDI children
- Tab controls
- TabbedMDIs
- Dockers
- Dock containers

Each of these use view windows in the same way.

#### **Customising the Frame's Creation**

The OnCreate function determines how the frame will be created. It is this function which creates the various child windows used by the frame, namely the status bar, toolbar, rebar, menu bar, and the view window. Override OnCreate if you wish to modify the creation of the various child windows.

There are several member variables used in OnCreate that affect the appearance of the frame. The following code is part of the generic starter for a frame application. The appearance of the frame can be altered by removing the comment from in front of one or more of the member variables listed.

```
void CMainFrame::OnCreate()
{
    // OnCreate controls the way the frame is created.
    // Overriding CFrame::Oncreate is optional.
    // The default for the following variables is TRUE
    // m_ShowIndicatorStatus = FALSE; // Don't show statusbar indicators
    // m_ShowMenuStatus = FALSE; // Don't show toolbar or menu statu
    // m_UseReBar = FALSE; // Don't use rebars
    // m_UseThemes = FALSE; // Don't use themes
    // call the base class function
    CFrame::OnCreate();
}
```

#### **Defining the Toolbar Buttons**

The bitmap image (or images) associated with the toolbar buttons is specified in the resource script file (Resource.rc). To modify the images displayed on the toolbar buttons, the bitmaps associated with the toolbar will need to be edited accordingly. If the programming environment doesn't include a resource editor for this purpose, a free resource editor such as <u>ResEdit</u> might prove useful. By default, the frame uses the bitmap resource associated with IDW\_MAIN when displaying the toolbar. The entry in Resource.rc that associates the bitmap with the resource ID looks like this.

IDW\_MAIN

BITMAP

"res\\toolbar.bmp"

The following code demonstrates how configure the commands issued by the toolbar when the button is pressed. This code should be placed in CMainFrame::SetupToolBar. This is an typical example SetupToolBar.

```
void CMainFrame::SetupToolBar()
{
  // Set the Resource IDs for the toolbar buttons
 AddToolBarButton( IDM FILE NEW
                                    );
  AddToolBarButton( IDM FILE OPEN
                                   );
 AddToolBarButton( IDM FILE SAVE
                                   );
 AddToolBarButton( 0 );
                                              // Separator
 AddToolBarButton( IDM_EDIT_CUT,
                                    FALSE ); // disabled button
 AddToolBarButton( IDM EDIT COPY,
                                    FALSE ); // disabled button
  AddToolBarButton( IDM EDIT PASTE, FALSE );
                                              // disabled button
 AddToolBarButton( 0 );
                                               // Separator
 AddToolBarButton( IDM FILE PRINT );
 AddToolBarButton( 0 );
                                               // Separator
  AddToolBarButton( IDM HELP ABOUT );
}
```

Refer to the tutorial for a demonstration of customising the toolbar buttons.

#### **Responding to Menu and Toolbar Input**

When a menu item is selected or a toolbar button is pressed, a WM\_COMMAND message is sent to the frame window. Override OnCommand to respond to these commands as demonstrated in the following code.

```
BOOL CMainFrame::OnCommand(WPARAM wParam, LPARAM /*lParam*/)
{
  // OnCommand responds to menu and and toolbar input
  switch(LOWORD(wParam))
  {
  case IDM FILE EXIT:
    // End the application when exit is selected from the menu
    PostQuitMessage(0);
    return TRUE;
  case IDM HELP ABOUT:
    // Display the help dialog
    OnHelp();
    return TRUE;
  }
  return FALSE;
}
```

#### **Customising the Toolbar**

Various aspects of the toolbar can be configured. The image list associated with normal, hot, and disabled toolbar buttons are set by the SetToolBarImages function. The following code demonstrates the use of the SetToolBarImages function, and shows how to add text to toolbar buttons.

```
void CMainFrame::SetupToolBar()
{
 // Set the Resource IDs for the toolbar buttons
 AddToolBarButton( IDM FILE NEW
                                   );
 AddToolBarButton( IDM FILE OPEN
                                   );
 AddToolBarButton( IDM FILE SAVE
                                   );
 AddToolBarButton( 0 );
                                              // Separator
 AddToolBarButton( IDM_EDIT_CUT,
                                    FALSE ); // disabled button
 AddToolBarButton( IDM EDIT COPY,
                                    FALSE ); // disabled button
 AddToolBarButton( IDM EDIT PASTE, FALSE );
                                             // disabled button
 AddToolBarButton( 0 );
                                              // Separator
 AddToolBarButton( IDM FILE PRINT );
 AddToolBarButton( 0 );
                                              // Separator
 AddToolBarButton( IDM HELP ABOUT );
 // Use larger buttons
 SetToolBarImages(RGB(192,192,192), IDB TOOLBAR NORM, IDB TOOLBAR HOT,
```

```
// Add some text to the buttons
 CToolBar& TB = GetToolBar();
 TB.SetButtonText(IDM FILE NEW,
                                   T("New"));
                                 _T("0pen"));
 TB.SetButtonText(IDM FILE OPEN,
 TB.SetButtonText(IDM_FILE_SAVE, _T("Save"));
 TB.SetButtonText(IDM EDIT CUT,
                                   T("Cut"));
                                  _T("Copy"));
 TB.SetButtonText(IDM_EDIT_COPY,
 TB.SetButtonText(IDM_EDIT_PASTE, _T("Paste"));
 TB.SetButtonText(IDM_FILE_PRINT, _T("Print"));
                                  _T("View Menu"));
 TB.SetButtonText(IDM VIEWMENU,
 TB.SetButtonText(IDM_HELP_ABOUT, T("About"));
}
```

#### **Customising Menu Items**

Menus are typically defined in the program's resource script (resource.rc). A menu definition might look like this:.

```
IDW MAIN MENU
BEGIN
    POPUP "&File;"
    BEGIN
        MENUITEM "New...",
                                                 IDM FILE NEW
        MENUITEM "&Open...;",
                                                  IDM FILE OPEN
        MENUITEM "&Save;",
                                                  IDM_FILE_SAVE
        MENUITEM "Save &As...;",
                                                  IDM FILE SAVEAS
        MENUITEM SEPARATOR
        MENUITEM "&Print;",
                                                  IDM_FILE_PRINT
        MENUITEM SEPARATOR
        MENUITEM "E&xit;",
                                                  IDM FILE EXIT
    END
    POPUP "&Edit;"
    BEGIN
        MENUITEM "Undo\tCtrl+Z",
                                                 IDM_EDIT_UNDO, GRAYED
        MENUITEM "Redo\tShift+Ctrl+Z",
                                                 IDM EDIT REDO, GRAYED
        MENUITEM SEPARATOR
        MENUITEM "Cut\tCtrl+X",
                                                 IDM EDIT CUT, GRAYED
        MENUITEM "Copy\tCtrl+C"
                                                 IDM EDIT COPY, GRAYED
        MENUITEM "Paste\tCtrl+V",
                                                 IDM EDIT PASTE, GRAYED
        MENUITEM "Delete\tDel",
                                                 IDM EDIT DELETE, GRAYED
    END
    POPUP "&View;"
    BEGIN
        MENUITEM "&Tool; Bar",
                                                  IDW VIEW TOOLBAR, CHECK
        MENUITEM "&Status; Bar",
                                                  IDW VIEW STATUSBAR, CHE
    END
    POPUP "&Help;"
    BEGIN
        MENUITEM "&About;",
                                                  IDM HELP ABOUT
```

#### END END

Menu items defined with the "GRAYED" flag are initially disabled. Menu items defined with the "CHECKED" flag are displayed with a check box.

Menu items can also be displayed with an image. The framework will automatically add images for those menu items defined with the same command ID as those used in the Toolbar. We can add icons for other menu items as well. The AddMenuIcons function adds a group of icons for menu items, and the AddMenuIcon function adds icons for individual menu items. Code like this can be added to CMainFrame::SetupToolBar to add icons for menu items.

```
// Add some extra icons for menu items
AddMenuIcon(IDM_FILE_NEWSIMPLE, GetApp()->LoadIcon(IDI_SIMPLE));
AddMenuIcon(IDM_FILE_NEWRECT, GetApp()->LoadIcon(IDI_RECT));
AddMenuIcon(IDM_FILE_NEWTEXT, GetApp()->LoadIcon(IDI_TEXT));
AddMenuIcon(IDM_FILE_NEWLIST, GetApp()->LoadIcon(IDI_FILEVIEW));
AddMenuIcon(IDM_FILE_NEWTREE, GetApp()->LoadIcon(IDI_CLASSVIEW));
```

#### Themes

Themes provide an opportunity to set the colour and styles of various parts of the frame, namely the menu, the toolbar and the rebar. By default, the frame will call the SetTheme function from inside OnCreate to set these various colours and styles. SetTheme can be overridden to modify these. This is example shows how set the theme to grey.

```
void CMainFrame::SetTheme()
{
    BOOL T = TRUE;
    BOOL F = FALSE;

    MenuTheme mt = {T, RGB(182, 189, 210), RGB(182, 189, 210), RGB(200, 1
    ReBarTheme rbt = {T, RGB(212, 208, 200), RGB(212, 208, 200), RGB(230,
    StatusBarTheme sbt = {T, RGB(212, 208, 200), RGB(212, 208, 200)};
    ToolBarTheme tbt = {T, RGB(182, 189, 210), RGB(182, 189, 210), RGB(133
    SetMenuTheme(&mt;); // Sets the theme for popup menus and MenuBar
    SetReBarTheme(&rbt;);
    SetStatusBarTheme(&sbt;);
    SetToolBarTheme(&tbt;);
}
```

The definition of the various theme structures are as follows:

```
struct ReBarTheme
{
   BOOL UseThemes; // TRUE if themes are used
   COLORREF clrBkgnd1; // Colour 1 for rebar background
   COLORREF clrBkgnd2; // Colour 2 for rebar background
   COLORREF clrBand1; // Colour 1 for rebar band background. Use NULL i
   COLORREF clrBand2; // Colour 2 for rebar band background. Use NULL i
```

```
// Bands are rendered with flat rather than raise
  BOOL FlatStyle;
  BOOL KeepBandsLeft; // TRUE if we always keep bands left
  BOOL LockMenuBand;
                           // Lock MenuBar's band up top, without gripper
                           // Use rounded band borders
  BOOL RoundBorders;
  BOOL ShortBands; // Allows bands to be shorter than maximum availa
BOOL UseLines: // Displays horizontal lines between bands
  BOOL UseLines;
                           // Displays horizontal lines between bands
};
struct MenuTheme
Ł
  BOOL UseThemes;
                            // TRUE if themes are used
  COLORREF clrHot1;
                            // Colour 1 for hot button
  COLORREF clrHot1; // Colour 1 for hot button
COLORREF clrHot2; // Colour 2 for hot button
  COLORREF clrPressed1; // Colour 1 for pressed button
  COLORREF clrPressed2; // Colour 2 for pressed button
  COLORREF clrOutline; // Colour for border outline
};
struct StatusBarTheme
{
  BOOL UseThemes; // TRUE if themes are used
COLORREF clrBkgnd1; // Colour 1 for statusbar background
  COLORREF clrBkgnd2; // Colour 2 for statusbar background
};
struct ToolBarTheme
  BOOL UseThemes; // TRUE if themes are used
COLORREF clrHot1; // Colour 1 for hot button
COLORREF clrHot2; // Colour 2 for hot button
  COLORREF clrPressed1; // Colour 1 for pressed button
  COLORREF clrPressed2; // Colour 2 for pressed button
  COLORREF clrOutline; // Colour for border outline
};
```

Refer to the themes sample application for a demonstration of the different sorts of effects that can be achieved by altering a frame's theme.

#### Using the Registry

Modern applications are expected to save their settings in the registry. To enable the loading and saving of program settings use the LoadRegistrySettings function in CMainFrame's constructor. If you also wish to load and save a list of Most Recently Used (MRU) files, call the LoadRegistryMRU function settings as well.

```
CMainFrame::CMainFrame()
{
   // Set m_View as the view window of the frame
   SetView(m_View);
```

```
// Set the registry key name, and load the initial window position
// Use a registry key name like "CompanyName\\Application"
LoadRegistrySettings(_T("Win32++\\Scribble Sample"));
// Load the settings from the registry with 4 MRU entries
LoadRegistryMRUSettings(4);
}
```

By default CFrame saves the frame size and position in the registry. To load and save other settings, override the LoadRegistyrSettings and SaveRegistrySettings functions in CMainFrame.

# **MDI Frames**

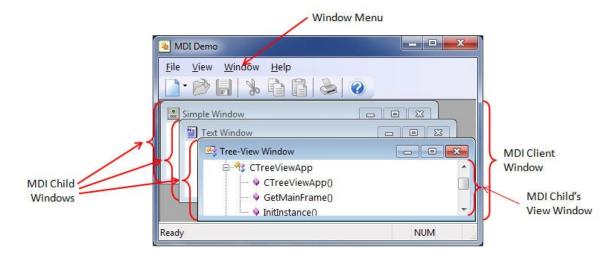
## Description

Multiple Document Interface (MDI) frames are capable of managing and displaying a number of MDI children. These MDI Children have their own view window.

Refer to the MDIFrame and MDIFrameDemo samples for demonstration of how to <u>CMDIFrame</u> and <u>CMDIChild</u> to create MDI applications.

#### **MDI Frame Components**

The following diagram illustrates a typical MDI frame application.



The toolbar and menubar are used to accept input from the user. The status bar is used to display status information.

#### **Defining the MDI Frame**

To define a MDI frame, inherit a class from <u>CMDIFrame</u> as shown below.

```
// MDI frames are inherrited from CMDIFrame
class CMainMDIFrame : public CMDIFrame
{
    public:
        CMainMDIFrame(void);
        virtual ~CMainMDIFrame();

protected:
    virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
    virtual int OnCreate(LPCREATESTRUCT pcs);
    virtual void OnInitialUpdate();
```

```
virtual void SetupToolBar();
virtual LRESULT WndProc(UINT uMsg, WPARAM wParam, LPARAM lParam);
};
```

Unlike SDI frames, we do not need to specify a view window for the frame. The Win32++ framework automatically assigns the MDI client window as the view window for the frame. The constructor for our MDI frame could look like this.

```
CMainMDIFrame::CMainMDIFrame()
{
    // Set the registry key name, and load the initial window position
    // Use a registry key name like "CompanyName\\Application"
    LoadRegistrySettings(_T("Win32++\\MDI Frame"));
}
```

#### **Defining a MDI Child**

To define a MDI child, inherit a class from <u>CMDIChild</u> as shown below.

```
// Declaration of CSimpleMDIChild
class CSimpleMDIChild : public CMDIChild
{
public:
   CSimpleMDIChild();
   virtual ~CSimpleMDIChild();
protected:
   virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);
   virtual void OnInitialUpdate();
   virtual LRESULT WndProc(UINT uMsg, WPARAM wParam, LPARAM lParam);
private:
   CSimpleView m_View;
};
```

#### The MDI Child's Menu and View

Normally each different type of MDI child has a different menu. The MDI Frame's menu is changed to the MDI child's menu when the MDI child becomes the active window. The menu for each MDI child is usually defined in the resource script file (resource.rc).

The view window for the MDI child is set in the same way we would set the view window for a SDI frame. The constructor for the class inherited from <u>CMDIChild</u> would typically look something like this.

```
CMDIChildSimple::CMDIChildSimple()
{
   SetView(m_View);
   HINSTANCE hResource = GetApp()->GetResourceHandle();
   HMENU hChildMenu = LoadMenu(hResource, _T("MdiMenuSimple"));
   SetHandles(hChildMenu, NULL);
```

### Adding a MDI child window

The AddMDIChild function is used to add a MDI child window. The following code demonstrates how to create a new MDI child window in response to a selection from the frame's menu or toolbar.

```
B00L CMainMDIFrame::OnCommand(WPARAM wParam, LPARAM /*lParam*/)
{
    switch (LOWORD(wParam))
    {
        case IDM_FILE_NEW:
            AddMDIChild(new CMDIChildView); // CMDIFrame deletes this pointer
            return TRUE;
    }
    return FALSE;
}
```

Note that the pointer created by "new" is deleted automatically by <u>CMDIFrame</u> when the MDI child window is destroyed.

}

# **Graphics Device Interface (GDI)**

### About the GDI

The Graphics Device Interface (GDI) is part of the Windows operating system. It provides applications with a means of sending graphics information to devices such as the video display and printer.

When using the GDI, the graphics device is represented as a device context (DC). All drawing calls are made through a device-context object, which encapsulates the Windows APIs for drawing lines, shapes, and text. Device contexts allow device-independent drawing in Windows. GDI objects such as bitmaps, brushes, palettes and pens are selected into the device context before they can be used to display the graphic information. Fonts can also be selected into the device context to manage the rendering of text.

#### The GDI classes in Win32++

The set of GDI classes provided by Win32++ are as follows:

- <u>CDC</u>
- <u>CClientDC</u>
- <u>CMemDC</u>
- <u>CMetaFileDC</u>
- <u>CPaintDC</u>
- <u>CWindowDC</u>
- <u>CGDIObject</u>
- <u>CBitmap</u>
- <u>CBrush</u>
- <u>CFont</u>
- <u>CPalette</u>
- <u>CPen</u>
- <u>CRgn</u>

The CDC classes (CDC, ClientDC, CMemDC, CMetaFileDC, CPaintDC, CWindowDC) provides a GDI device context. The CDC classes can also be used to create the other GDI resources such as Bitmaps, Brushes, Fonts etc. These are automatically selected into the device context when they are created. These are also deleted when the CDC goes out of scope.

With their ability to create device contexts as well as other GDI resources, the CDC classes are sufficient for most GDI programming needs. Sometimes however, we need to have the GDI resources separated from the device context. Wrapper classes for GDI resources are provided for this purpose. These classes are CBitmap, CBrush, CFont, CPalette, CPen and CRgn.

#### Using the GDI classes

#### Using the Windows API without CDC

This code demonstrates how to use the Windows API functions directly to draw a line with the GDI.

```
void DrawLine()
{
  HDC hdcClient = ::GetDC(m hWnd);
  HDC hdcMem = ::CreateCompatibleDC(hdcClient);
  HBITMAP hBitmap = ::CreateCompatibleBitmap(hdcClient, cx, cy);
  HBITMAP hOldBitmap = (HBITMAP)::SelectObject(hdcMem, hBitmap);
  HPEN hPen = ::CreatePen(PS SOLID, 1, RGB(255,0,0);
  HPEN hOldPen = (HPEN)::SelectObject(hdcMem, hPen);
  ::MoveToEx(hdcMem, 0, 0, NULL);
  ::LineTo(hdcMem, 50, 50);
  ::BitBlt(hdcClient, 0, 0, cx, cy, hdcMem, 0, 0);
  ::SelectObject(hdcMem, hOldPen);
  ::DeleteObject(hPen);
  ::SelectObject(hdcMem, hOldBitmap);
  ::DeleteObject(hBitmap);
  ::DeleteDC(hdcMem);
  ::ReleaseDC(m hWnd, hdcClient);
}
```

## Using CDC classes alone

This code performs the same task as shown above using the CDC class.

```
void DrawLine()
{
    CClientDC dcClient(this)
    CMemDC dcMem(&dcClient;);
    dcMem.CreateCompatibleBitmap(&dcClient;, cx, cy);
    CMemDC.CreatePen(PS_SOLID, 1, RGB(255,0,0);
    CMemDC.MoveTo(0, 0);
    CMemDC.LineTo(50, 50);
    dcClient.BitBlt(0, 0, cx, cy, &CMemDC;, 0, 0);
}
```

## Using CDC classes with CPen

This code uses a CDC and a separate CPen to draw the line.

```
void DrawLine()
{
    CClientDC dcClient(this)
    CMemDC dcMem(&dcClient;);
    dcMem.CreateCompatibleBitmap(&dcClient;, cx, cy);
    CPen MyPen(PS_SOLID, 1, RGB(255,0,0));
    CPen* pOldPen = dcMem.SelectObject(&MyPen;);
    dcMem.MoveTo(0, 0);
    dcMem.LineTo(50, 50);
    dcClient.BitBlt(0, 0, cx, cy, &dcMem;, 0, 0);
```

Notes:

- When the CDC object drops out of scope, it's destructor is called, cleaning up any GDI resources it created, as well as the device context.
- The device context is returned to its initial state before deletion, removing the need to select the OldPen back into the device context.
- When the CPen object drops out of scope, it's destructor is called, deleting its associated GDI object (HPEN).

#### **Creating a Window DC**

A window DC is a device context which represents the entire window, including the non-client area. We can create a window DC as follows:

#### CWindowDC dcWindow(this);

#### **Creating a Window Client DC**

The following code creates a device context for the client area of the window. The following code shows how we can create a DC for the window's client area.

```
CClientDC dcClient(this);
```

#### Creating a Memory DC

As the name suggests, a memory DC is created in memory. Memory DCs are typically used for double buffering. With this programming technique we perform the drawing tasks to a memory DC, and then copy the bitmap to our window.

When creating a memory DC we provide a pointer to the CWindowDC or CClientDC that the memory DC is compatible with. We then create the compatible bitmap that the memory DC will draw to. The following code demonstrates how this is done.

```
// Create our memory client DC
CRect rc = GetClientRect();
CClientDC dcClient(this);
// Create our memory DC and compatible bitmap
CMemDC dcMem(&dcClient);
dcMem.CreateCompatibleBitmap(&dcClient, rc.Width(), rc.Height());
// Draw some stuff on the memory DC
dcMemC.CreatePen(PS_SOLID, 1, RGB(255,0,0);
dcMem.MoveTo(0, 0);
dcMem.LineTo(50, 50);
// Copy the Memory DC's bitmap to the window's client DC
```

```
}
```

```
dcClient.BitBlt(0, 0, rc.Width(), rc.Height(), &dcMem, 0, 0, SRCCOPY);
```

The use of a memory DC doesn't necessarily make the graphics any faster, but can reduce or even eliminate annoying flicker.

### Handling OnDraw and OnEraseBkgnd

The OnDraw and OnEraseBkgnd functions provide a pointer to the CDC for us to use. The following code demonstrates how these functions can be used.

```
void CView::OnDraw(CDC* pDC)
{
    CRect rc = GetClientRect();
    // Centre some text in our view window
    pDC->DrawText(_T("View Window"), -1, rc, DT_CENTER | DT_VCENTER | DT_S
}
BOOL CView::OnEraseBkgnd(CDC* pDC)
{
    CBrush MyBrush(RGB(255,255,230)); // this could also be a member varia
    CRect rc = GetClientRect();
    FillRect(rc, MyBrush);
}
```

Notes:

- A device context assigned to a CDC object will be released or deleted when the CDC is destroyed, unless it is detached.
- A GDI resource created by one of the CDC member functions will be deleted when the CDC object is destroyed.
- The GDI resources selected into the CDC with SelectObject are not deleted when the CDC goes out of scope. Only GDI resources created by the CDC are automatically deleted.
- GDI resources belonging to the various GDI wrapper classes (eg. CPen) are automatically deleted when the class object is destroyed, unless they are detached .
- A bitmap GDI object can only be selected into one device context at a time.
- Set the region's shape before selecting it into a DC.

# Docking

# Description

The <u>CDocker</u> class adds both splitter windows and docking to the Win32++ framework. <u>CFrame</u> is inherited from <u>CDocker</u>, so frames and MDI frames support docking directly.

Splitter windows have a moveable splitter bar between the windows. Windows which have docking enabled have the moveable splitter bar too, but they can also be dragged away from the view, or undocked. Undocked docking windows can be dragged over the view window and docked.

🔠 Splitter Example	
<u>File E</u> dit <u>V</u> iew <u>H</u> elp	
	Text Edit Window
Simple View	You can type some text here
TreeView  CTreeViewApp  CTreeViewApp()  GetMainFrame()  CMainFrame	Name Size ListView Main.cpp 1 KB ListViewApp.cpp 3 KB ListViewApp.h 1 KB KB
Ready	NUM

The <u>CDocker</u> class creates specialized windows called "Dockers" capable of allowing other Dockers to dock to and undock from them. When docking, the undocked Docker is dragged over another Docker. Various visual clues such has the dock targeting (small arrow-like images), and the dock hinting (where a portion of the destination window turns blue) provide a hint as to where the Docker will be docked. To facilitate undocking, the caption of the docked window is dragged and dropped.

Every Docker has a view window. These views can be any resizable child window, and are set in the same way as views for Frames and MDI children. DockContainers (provided by the <u>CDockContainer</u> class) are a specialized view which add additional docking features when used as the view window for a Docker. DockContainers are decribed in more detail below.

A Docker which is docked within another Docker is said to be a "dock child" of that Docker. There is no theoretical limit to how many dock children a Docker may have. There is also no theoretical limit as to the depth of the child/parent relationship. That's to say there can be any number of dock children within dock

children within dock children etc.

The primary or first Docker is referred to as the Dock Ancestor. This would typically be the frame or MDI frame window. Other Dockers are added to this dock hierarchy using the AddDockedChild or AddUndockedChild functions. This group of Dockers is said to be "related" or in the same docking group. These Dockers can dock and undock from each other.

A Docker will dock to any one of the four sides of another related Docker (left, right, top or bottom). When a Docker is docked within a child Docker, this is referred to as "inner docking". When a Docker is docked to the side of the Dock Ancestor, this is referred to as "outer docking". Outer docking and inner docking have different dock targeting visual cues.

#### **Docker Components**

The following image illustrates some of the components of Docker windows. In this case the window being docked is dragged over a Docker with a Container view. Hence we see the center target option within the Inner Dock Targets for Container within Container docking.

#### Setting the Docker's view

The following code shows how the Docker's view is assigned, and how to set the width of the splitter bar.

```
CDockClasses::CDockClasses()
{
   SetView(m Classes);
```

```
// Set the width of the splitter bar
SetBarWidth(8);
}
```

Dockers can have a number of styles. These styles can determine where the Docker is docked, whether or not it can be undocked, and where dock children are permitted to dock. The dock style is initially specified when the Docker is added to the docking hierarchy. The complete set of dock styles are as follows:

DS_DOCKED_LEFT	Dock the child left
DS_DOCKED_RIGHT	Dock the child right
DS_DOCKED_TOP	Dock the child top
DS_DOCKED_BOTTOM	Dock the child bottom
DS_NO_DOCKCHILD_LEFT	Prevent a child docking left
DS_NO_DOCKCHILD_RIGHT	Prevent a child docking right
DS_NO_DOCKCHILD_TOP	Prevent a child docking at the top
DS_NO_DOCKCHILD_BOTTOM	Prevent a child docking at the bottom
DS_NO_RESIZE	Prevent resizing
DS_NO_CAPTION	Prevent display of caption when docked
DS_NO_CLOSE	Prevent closing of a docker while docked
DS_NO_UNDOCK	Prevent undocking of the docker
DS_CLIENTEDGE	Has a 3D border when docked
DS_NO_FIXED_RESIZE	Resize dock children proportionally. Unless set, dock children have a fixed size when the dock parent is resized.
DS_DOCKED_CONTAINER	Dock a container within a container
DS_DOCKED_LEFTMOST	Leftmost outer docking
DS_DOCKED_RIGHTMOST	Rightmost outer docking
DS_DOCKED_TOPMOST	Topmost outer docking
DS_DOCKED_BOTTOMMOST	Bottommost outer docking

#### Adding Dock Children

The following code shows how to add dock children. Note that the newly added Docker can be a dock child of the dock ancestor or a dock child of any other Docker descendant of the to the dock ancestor.

```
void CMainFrame::LoadDefaultDockers()
{
    // Note: The DockIDs are used for saving/restoring the dockers state
```

```
DWORD dwStyle = DS_CLIENTEDGE; // The style added to each docker
// Add the parent dockers
CDocker* pDockRight = AddDockedChild(new CDockClasses, DS_DOCKED_RIGH
CDocker* pDockBottom = AddDockedChild(new CDockText, DS_DOCKED_BOTTOM
// Add the remaining dockers
pDockRight->AddDockedChild(new CDockFiles, DS_DOCKED_CONTAINER | dwSty
pDockRight->AddDockedChild(new CDockClasses, DS_DOCKED_CONTAINER | dwSty
pDockRight->AddDockedChild(new CDockFiles, DS_DOCKED_CONTAINER | dwSty
pDockRight->AddDockedChild(new CDockFiles, DS_DOCKED_CONTAINER | dwSty
pDockBottom->AddDockedChild(new CDockOutput, DS_DOCKED_CONTAINER | dwSty
pDockBottom->AddDockedChild(new CDockText, DS_DOCKED_CONTAINER | dwSty
pDockBottom->AddDockedChild(new CDockOutput, DS_DOCKED_CONTAINER | dwSty
```

## Saving the Dockers in the Registry

}

CDocker provides built in support for saving the Dockers in the registry. The functions which perform this task are CDocker::LoadDockRegistrySettings and CDocker::SaveDockRegistrySettings.

The first step in using these functions is to override CDocker::NewDockerFromID for the Dock Ancestor. This function can create a new Docker from the specified Docker ID. In this case CDockSimple is the CDocker class for our Dock Ancestor, so the NewDockerFromID function looks like this:

```
CDocker* CMainFrame::NewDockerFromID(int nID)
{
  CDocker* pDock = NULL;
  switch(nID)
  Ł
  case ID DOCK CLASSES1:
    pDock = new CDockClasses;
    break;
  case ID DOCK CLASSES2:
    pDock = new CDockClasses;
    break;
  case ID DOCK FILES1:
    pDock = new CDockFiles;
    break;
  case ID DOCK FILES2:
    pDock = new CDockFiles;
    break;
  case ID DOCK OUTPUT1:
    pDock = new CDockOutput;
    break;
  case ID DOCK OUTPUT2:
    pDock = new CDockOutput;
    break:
  case ID DOCK TEXT1:
```

```
pDock = new CDockText;
break;
case ID_DOCK_TEXT2:
    pDock = new CDockText;
    break;
default:
    TRACE(_T("Unknown Dock ID\n"));
    break;
}
return pDock;
}
```

All that remains now is to use the CDocker's LoadDockRegistrySettings and SaveDockRegistrySettings in CMainFrame. The following code demonstrates how this might be done.

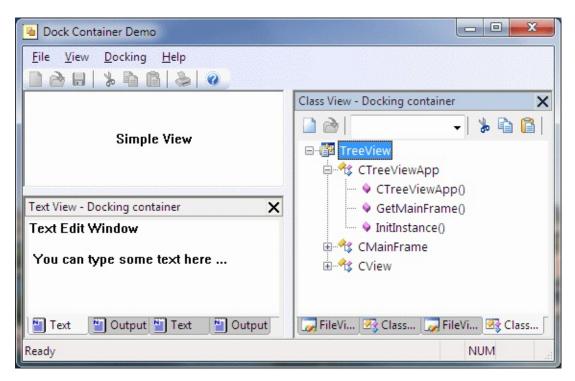
```
void CMainFrame::OnInitialUpdate()
{
    m_DockView.SetDockStyle(DS_CLIENTEDGE);
    // Load dock settings
    if (!LoadDockRegistrySettings(GetRegistryKeyName()))
        LoadDefaultDockers();
    // PreCreate initially set the window as invisible, so show it now.
    ShowWindow();
}
void CMainFrame::SaveRegistrySettings()
{
    CFrame::SaveRegistrySettings();
    SaveDockRegistrySettings(GetRegistryKeyName());
}
```

Refer to the DockSimple, DockContainer and DockTabbedMDI samples for a demonstration of docking.

#### **Dock Containers**

DockContainers are a specialized view window for Dockers. It is a tab control which has been designed to co-operate with docking. While Dockers allows any child window to be used as the view window, DockContainers add additional features when used as the view window for Dockers. These additional features include container within container docking (where the newly docked DockContainer adds another tab), as well as additions to the dock targeting and dock hinting visual cues. DockContainers can also have a toolbar. The use of a toolbar is optional, but when used they are set up in the same way as toolbars for Frames.

The <u>CDockContainer</u> class adds DockContainers to the Win32++ framework. Each DockContainer has a single view window of its own. This view window can be any child window, and is set in the same way as views for Frames and MDI children.



The various attributes of the DockContainer are typically set in its constructor. The following code demonstrates how to specify a DockContainer's tab text, tab icon, view window, and the text which will be displayed in the docked caption.

```
CContainClasses::CContainClasses()
{
   // Note: CContainClasses inherits from CDockContainer
   SetTabText(_T("ClassView"));
   SetTabIcon(IDI_CLASSVIEW);
   SetDockCaption (_T("Class View - Docking container"));
   SetView(m_ViewClasses);
}
```

If the DockContainer has a toolbar, it is set up in the same way as the toolbar for a Frame. The following code demonstrates how the SetupToolBar function is used to specify the ToolBar's bitmap and resource IDs, as well as how to configure the toolbar themes to match those of the Frame.

```
void CContainClasses::SetupToolBar()
{
    // Set the Bitmap resource for the toolbar
    GetToolBar().SetImages(RGB(192,192,192), IDW_MAIN, 0, 0);
    // Set the Resource IDs for the toolbar buttons
    AddToolBarButton( IDM_FILE_NEW );
    AddToolBarButton( IDM_FILE_OPEN, FALSE );
    AddToolBarButton( 0 ); // Separator
    AddToolBarButton( IDM_FILE_SAVE, FALSE );
```

```
AddToolBarButton( 0 );
                            // Separator
AddToolBarButton( IDM EDIT CUT
                                      );
AddToolBarButton( IDM EDIT COPY
                                      );
AddToolBarButton( IDM EDIT PASTE
                                      );
AddToolBarButton( 0 );
                            // Separator
AddToolBarButton( IDM FILE PRINT, FALSE );
AddToolBarButton( 0 );
                             // Separator
AddToolBarButton( IDM HELP ABOUT
                                      );
// Add the ComboBarEx control to the toolbar
AddCombo();
```

}

The following example demonstrates how to add DockContainers to an exiting Docker. Note that the DS\_DOCKED\_CONTAINER style is used to dock a container within a container. Only DockContainers support this style.

```
void CMainFrame::LoadDefaultDockers()
{
 // Note: The DockIDs are used for saving/restoring the dockers state
 // Note: CDockClasses, CDockFiles, CDockOutput and CDockText inherit f
           have a view which inherits from CDockContainer.
 DWORD dwStyle = DS CLIENTEDGE; // The style added to each docker
 // Add the parent dockers
 CDocker* pDockRight = AddDockedChild(new CDockClasses, DS DOCKED RIGH
 CDocker* pDockBottom = AddDockedChild(new CDockText, DS DOCKED BOTTOM
 // Add the remaining dockers
 pDockRight->AddDockedChild(new CDockFiles, DS DOCKED CONTAINER | dwSty
  pDockRight->AddDockedChild(new CDockClasses, DS_DOCKED_CONTAINER | dwS
 pDockRight->AddDockedChild(new CDockFiles, DS_DOCKED_CONTAINER | dwSty
 pDockBottom->AddDockedChild(new CDockOutput, DS DOCKED CONTAINER | dwS
 pDockBottom->AddDockedChild(new CDockText, DS DOCKED CONTAINER | dwSty
 pDockBottom->AddDockedChild(new CDockOutput, DS DOCKED CONTAINER | dwS
}
```

## **Text Conversion**

#### Introduction

There are several different types of text strings used in windows programming. These include:

#### ANSI

ANSI is an acronym for the American National Standards Institute. In windows programming, the term ANSI usually refers to single-byte ISO-8859 encodings. ANSI text strings are stored as an array of single byte char (or CHAR).

#### BSTR

BSTR is an abbreviation of Basic String (or perhaps Binary String). It was originally used in Visual Basic, and is often used in COM programming today. A BSTR string is a specialized Unicode string. It begins with a length field, which is followed by a string of Unicode characters. A BSTR is always NULL terminated, but can also contain NULL characters within the string.

#### OLE

Microsoft introduced a number of macros with "OLE" in the name, such as LPOLESTR. The term as it relates to text strings is largely superseded by Unicode (i.e. Wide) strings, however there are still a few Windows API functions that take LPOLESTR arguments. In modern Windows programming, OLE strings are Unicode strings.

#### TCHAR

TCHAR text strings are Unicode if the UNICODE macro is defined, otherwise they are ANSI. TCHAR is the type of text string used in most Windows API functions. It helps developers to produce code capable of supporting both ANSI and Unicode text strings.

#### Wide or Unicode

Unicode is an international standard for the representation of text characters. Unlike ANSI which uses a single byte for each character, Unicode (in Windows) uses two bytes for each character. Unicode text strings are stored as an array of wchar (or WCHAR). Modern Microsoft Windows operating systems (since Windows ME) have standardised on Unicode.

#### String Conversion Functions in Win32++

Win32++ contains the following functions to convert from one string type to another.

A2BSTR	Convert from ANSI to BSTR
A2OLE	Convert from ANSI to OLE
A2T	Convert from ANSI to TCHAR
A2W	Convert from ANSI to WCHAR

OLE2AConvert from OLE to ANSIOLE2TConvert from OLE to TCHAR

OLE2W Convert from OLE to WCHAR T2A Convert from TCHAR to ANSI T2BSTR Convert from TCHAR to BSTR T2OLE Convert from TCHAR to OLE T2W Convert from TCHAR to WCHAR W2A Convert from WCHAR to ANSI W2BSTR Convert from WCHAR to BSTR W2OLE Convert from WCHAR to OLE W2OLE Convert from WCHAR to OLE

#### **Using the Text Conversion Functions**

Each of the text conversion functions described above are actually typedefs of text conversion classes such as CA2W and CW2A. They each return a pointer to the appropriate text string type. These text conversions can be used as either classes or functions. When using them as functions it is important to remember that the returned pointer goes out of scope immediately. This means the returned pointer cannot be saved for later use. The following examples illustrate this.

#### Example 1 - Using the text converter as a class

In this example W2A is used as a class. A W2A object is created using its constructor, and this object remains in scope until the example function ends.

```
void ExampleFunctionW( LPCWSTR pWide )
{
   // Use the converion class as a class.
   W2A MyAnsiString( pWide );
   // MyAnsiString works like an LPCSTR (pointer to const CHAR), and can
   SetWindowTextA( MyAnsiString );   // The ANSI version of SetWindow
   // Note: MyAnsiString remains in scope until the function ends.
}
```

#### **Example 2** - Using the text converter as as a function.

In this example the result of the W2A text conversion is used immediately. The destructor of W2A is called after the call to SetWindowTextA.

```
void ExampleFunctionW( LPCWSTR pWide)
{
    // Convert from Wide (Unicode) to ANSI and use the result.
    SetWindowTextA( W2A( pWide ) ); // The ANSI version of SetWindowTex
```

#### **Example 3 - Storing a result for later use.**

This examples demonstrates some correct techniques for storing our text conversion result for later use. Note that we cannot simply store the pointer because it goes out of scope immediately.

```
void ExampleFunctionW( LPCWSTR pWide )
{
   // Store the result in a std::string
   std::string str = W2A( pWide ); // or std::string str = (LPCTSTR)W2A(
   // Store the result in an array
   char szArray[80];
   strcpy( szArray, W2A( pWide ) );
   // Store the result in a vector
   std::vector<char> MyAnsiVector(strlen(W2A(pWide))+1, '\0');
   strcpy(&MyAnsiVector.front(), W2A(pWide);
   // Do something with our stored string conversion
   // ...
}
```

#### **Example 4 - Incorrect use of the text conversion function.**

In this example the destructor for W2A is called before the call to SetWindowTextA. As a result, the contents of pAnsi are destroyed and the pointer no longer points to a valid character array.

```
void ExampleFunctionW( LPCWSTR pWide )
{
   // THIS IS INCORRECT!
   LPCSTR pAnsi = W2A( pWide );
   // W2A's destructor has already been called, so pAnsi is no longer poi
   SetWindowTextA( pAnsi ); // Behaviour of this line is undefined!
}
```

}

# CString

#### Introduction

CStrings are used for text strings. They can prove easier and safer to use than TCHAR arrays. CStrings are easily copied and modified, and handle null termination automatically.

The <u>CString</u> class provided with Win32++ is designed to behave in much the same way as CStrings provided with other frameworks like MFC and ATL.

#### **Assigning CStrings**

There are several ways to assign a CString. These include:

```
CString str1 = _T("Text string");
CString str2(_T("Text string"));
TCHAR szText[80] = _T("Text String");
CString str3 = szText;
CString str4 = _T('T'); // a single character
CString str5 = str1; // copy a CString
CString str6 = str1 + str2; // Concatenate two strings
// Assign a string like a c-style printf
CString str7;
str7.Format(_T("There are %d apples"), 5);
```

#### **Modifying CStrings**

The <u>CString</u> class has several member functions for modifying the contents of a CString. These include:

- Insert can be used to insert characters into the string.
- Delete can be used to remove characters from the string.
- MakeUpper and MakeLower converts the string to upper or lower case.
- Trim, TrimLeft and TrimRight can trim characters from the CString.
- Remove the specified character from the string.
- Replace an old sub-string with a new one.
- SetAt changes the character at the specified index.
- Truncate reduces the length of the string to the specified amount.

Coding Example:

```
// insert the word "sat" into the string
CString str("The cat on the mat");
str.Insert(8, _T("sat "));
```

// Convert the string to upper case
str.MakeUpper();

#### Accessing elements of a CString

Parts of a <u>CString</u> can be accessed in several different ways:

- Left, Mid and Right can be used to extract characters from the string.
- GetAt retrieves the character at the specified location.

Coding Example:

```
CString strl("The cat sat on the mat");
```

```
// Copy 3 characters, beginning at index 4 to str2
CString str2 = str1.Mid(4, 3);
assert(str2 == T("cat");
```

#### Finding elements in the CString

The following functions can be used to find the index of elements in the <u>CString</u>:

- Find and ReverseFind
- FindOneOf

Coding Example:

```
CString str( "The cat sat on the mat" );
int i = str.Find(_T("cat"));
assert(i == 4);
```

#### Using GetBuffer and ReleaseBuffer

<u>CString</u> can provide a pointer to an internal buffer. This allows a CString to be used in places where we would write to a character array.

Coding Example:

Here we use GetBuffer to allocate a buffer for use by the GetWindowText function.

```
int nLength = ::GetWindowTextLength(m_hWnd);
CString str;
::GetWindowText(m_hWnd, str.GetBuffer(nLength), nLength+1);
```

```
str.ReleaseBuffer();
```

Note: We must call ReleaseBuffer when we have finished writing to the buffer. This copies the contents of the buffer into the CString and frees the allocated buffer.

## **Library Reference**

## Classes

CAnimation A class used to create an animation control.

<u>CBitmap</u> A class used to create a bitmap resource. <u>CBitmapInfoPtr</u> A class used to create the BITMAPINFO structure. CBrush A class used to create a brush resource. CButton A class used to create a button control. CClientDC The class used to device contect for the client area of the window. CCmdBar The class used to create a command bar control. (Windows CE only). CComboBox A class used to create a combo box control. CComboBoxEx A class used to create a ComboBoxEx control. CContainer A class used for docking tabbed window. CCriticalSection The class used for thread synchronisation. CDateTime This class is used to create a date and time picker control. <u>CDC</u> This class is used to simplify working with device contexts and GDI graphics. <u>CDialog</u> The class responsible for creating modal and modeless dialogs. It can also be used to create dialog applications. <u>CDockContainer</u> The class is used to create docking containers. <u>CDocker</u> The class used for a docking window. Docking windows can also be used as splitter windows. CEdit A class used to create an edit control. CFile A class used to read from and write to files. CFont A class used to create a font resource. CFrame The class used to create a frame window. The frame window has a menu, toolbar, and a status bar. The client area of the frame window is occupied by a separate CWnd window, often called the view window. <u>CGDIObject</u> The base class for GDI objects including CBitmap, CBrush, CFont, CPalette, CPen, and CRgn. <u>CHeader</u> A class used to create a header control. CHotKey A class used to create a hot key control. CImageList A class used to create and manage Image Lists. <u>CIPAddress</u> The class used to create an IP address control. <u>CListBox</u> A class used to create a list-box control. <u>CListView</u> A class that is used to create a List-View control. <u>CMDIChild</u> The class for MDI child windows. All MDI children should be inherited from this class. CMDIFrame The class for MDI frames. <u>CMemDC</u> The class used to create memory device context. <u>CMenu</u> The class used to create and modify menus. CMenuBar The class used to create the menubar. A menubar is a specialized toolbar which mimics the behaviour of a menu. It is usually used within a rebar control. <u>CMetaFileDC</u> A class used to create metafile device context. <u>CMonthCalendar</u> A class used to create a month calendar control. <u>CPaintDC</u> A class used to create device context for the WM\_PAINT message. CPalette A class used to create a palette resource. <u>CPen</u> A class used to create a pen resource. <u>CPoint</u> A class that can be used in place of a POINT structure. <u>CProgressBar</u> A class used to create a progress bar control. <u>CPropertyPage</u> The class used to create a property page. A property sheet has one or more property pages. <u>CPropertySheet</u> The class used to create a property sheet. <u>CReBar</u> The class used to create a rebar control. <u>CRect</u> A class that can be used in place of a RECT structure. CRgn A class used to create a region. CRibbon The class used to add a Window 7 ribbon framework to a window. CRibbonFrame The class used to create a frame window with a Window 7 ribbon framework. A menu and toolbar will be used if the operating system doesn't support the ribbon. CRichEdit A class used to create a rich edit control. CScrollBar A class used to create a scroll bar control. CSize A class that can be used in place of a SIZE structure. CSlider The class used to create a slider control (sometimes referred to as a track bar control). CSocket The class used for networking. CSpinButton The

class used to create a spin button control (sometimes referred to as an up down control). <u>CStatic</u> The class used to create a static control. <u>CStatusBar</u> The class used to create a status bar. <u>CString</u> The class used to create and modify character strings. <u>CTab</u> The class used to create a Tab control. <u>CTabbedMDI</u> The class used for Tabbed MDI window. <u>CTaskDialog</u> The class used for to create a Task Dialog. <u>CToolBar</u> The class used to create a rebar control. <u>CToolTip</u> The class used to create a tool tip control. <u>CTreeView</u> A class used to create a Tree-View control. <u>CWceFrame</u> The class used to create a frame window on Pocket PCs. (Windows CE only). <u>CWebBrowser</u> The class used to create a web browser in a window. <u>CWinApp</u> The class responsible for initializing Win32++, and also provides the message loop. You inherit from this class to start the application. <u>CWindowDC</u> The class used to create a device context for the entire window, including the non-client area. <u>CWinException</u> A class which handles exceptions. <u>CWinThread</u> The class used to to create worker threads and GUI threads. A window can run in a GUI thread. <u>CWnd</u> The CWnd class represents a window. CWnd handles such things as window creation and window destruction, as well as determining how the window messages are handled. <u>Shared Ptr</u> A smart pointer that can be used in a STL container such as a vector.

#### **Global Functions**

Defined in winutils.h		
GetApp	CWinApp* GetApp();	
	Returns a pointer to the CWinApp derrived class.	
GetCursorPos	CPoint GetCursorPos();	
	Returns a CPoint containing the cursor position.	
GetComCtlVersion	<pre>int GetComCtlVersion();</pre>	
	Returns an integer which indicates the version of of the ComCtl32.dll used.	
GetSizeofMenuItemInfo	<pre>UINT GetSizeofMenuItemInfo();</pre>	
	Returns the correct size of the MENUITEMINFO structure for all versions of w	
GetSizeofNonClientMetrics	<pre>UINT GetSizeofNonClientMetrics();</pre>	
	Returns the correct size of the NONCLIENTMETRICS structure for all version	
GetWinVersion	<pre>int GetWinVersion();</pre>	
	Returns an integer indicating the version of windows running.	
IsAeroThemed	BOOL IsAeroThemed();	

	Returns TRUE if Aero themes are being used.	
IsXPThemed	BOOL IsXPThemed();	
	Returns TRUE if XP themes are being used.	
IsLeftButtonDown	B00L IsLeftButtonDown();	
ISECTEDUITOIDOWI	Returns TRUE if the left mouse button is pressed.	
lstrcpyn	LPTSTR lstrcpyn(LPTSTR lpstrDest, LPCTSTR lpstrSrc,	
louepyn	Copies a specified number of characters from a source string to a destination	
LoadCommonControls	<pre>void LoadCommonControls();</pre>	
	Registers and initializes certain common control window classes.	
	<pre>void TRACE(LPCSTR str);</pre>	
TRACE	<pre>void TRACE(LPCWSTR str);</pre>	
	Sends a string to the debug/output pane, or an external debugger.	
String Conversion Func	tions	
A2BSTR	Converts an ANSI string to a BSTR string.	
A2OLE	Converts an ANSI string to an OLE string.	
A2T	Converts an ANSI string to a TCHAR string.	
A2W	Converts an ANSI string to a WCHAR string.	
OLE2A	Converts an OLE string to an ANSI string.	
OLE2T	Converts an OLE string to a TCHAR string.	
OLE2W	Converts an OLE string to a WCHAR string.	
T2A	Converts a TCHAR string to an ANSI string.	
T2BSTR	Converts a TCHAR string to a BSTR string.	
T2OLE	Converts a TCHAR string to an OLE string.	
T2W	Converts a TCHAR string to a WCHAR string.	
W2A	Converts a WCHAR string to an ANSI string.	
W2BSTR	Converts a WCHAR string to a BSTR string.	
W2OLE	Converts a WCHAR string to an OLE string.	
W2T	Converts a WCHAR string to a TCHAR string.	
Defined in cstring.h		
LoadString	CString LoadString(UINT nID);	
-	Retrieves a string resource and returns the result as a CString.	

GetPidlLength	<pre>UINT GetPidlLength(LPITEMIDLIST pidl);</pre>
	Returns the length of the Item Identifier List. Item Identifier Lists identify the n folders within the shell namespace.

## Macros

GET_X_LPARAM	Retrieves the signed x-coordinate from the given LPARAM value.
GET_Y_LPARAM	Retrieves the signed y-coordinate from the given LPARAM value.
MAX	Returns the higher of two values.
MIN	Returns the lower of two values.
NO_USING_NAMESPACE	Define this macro to suppress using the Win32xx namespace.

## **Type declarations**

tString	<pre>typedef std::basic_string<tchar> tString;</tchar></pre>
tStringStream	<pre>typedef std::basic_stringstream<tchar> tStringStream;</tchar></pre>
BitmapPtr	typedef Shared_Ptr <cbitmap> BitmapPtr;</cbitmap>
BrushPtr	typedef Shared_Ptr <cbrush> BrushPtr;</cbrush>
DCPtr	typedef Shared_Ptr <cdc> DCPtr;</cdc>
GDIPtr	typedef Shared_Ptr <cgdiobject> GDIPtr;</cgdiobject>
FontPtr	typedef Shared_Ptr <cfont> FontPtr;</cfont>
ImageListPtr	typedef Shared_Ptr <cimagelist> ImageListPtr;</cimagelist>
MenuPtr	typedef Shared_Ptr <cmenu> MenuPtr;</cmenu>
PalettePtr	typedef Shared_Ptr <cpalette> PalettePtr;</cpalette>
PenPtr	typedef Shared_Ptr <cpen> PenPtr;</cpen>
RgnPtr	typedef Shared_Ptr <crgn> RgnPtr;</crgn>
WndPtr	typedef Shared_Ptr <cwnd> WndPtr;</cwnd>

## Messages

UWM_CLEANUPTEMPS	const UINT UWM_CLEANUPTEMPS = RegisterWindowMessa Message - posted to cleanup temporary CDCs, CWnds etc.
UWM_DOCKACTIVATE	<pre>#define UWM_DOCKACTIVATE (WM_APP + 0x3F01) Message - sent to dock ancestor when a docker is activated or deactivated</pre>
UWM_DOCKDESTROYED	<pre>#define UWM_DOCKDESTROYED (WM_APP + 0x3F02) Message - posted when docker is destroyed.</pre>
UWM_DRAWRBBKGND	<pre>#define UWM_DRAWRBBKGND (WM_APP + 0x3F03)</pre>

	Message - sent by rebar to parent to perform background drawing. Retu
UWM_DRAWSBBKGND	#define UWM_DRAWSBBKGND (WM_APP + 0x3F04)
	Message - sent by statusbar to parent to perform background drawing.
UWM_GETFRAMEVIEW	#define UWM_GETFRAMEVIEW (WM_APP + 0x3F05)
	Message - returns the HWND of the frame's view window.
UWM_GETMBTHEME	<pre>#define UWM_GETMBTHEME (WM_APP + 0x3F06)</pre>
	Message - returns a pointer to MenuBarTheme.
UWM GETRBTHEME	#define UWM_GETRBTHEME (WM_APP + 0x3F07)
	Message - returns a pointer to ReBarTheme.
UWM GETSBTHEME	<pre>#define UWM_GETSBTHEME (WM_APP + 0x3F08)</pre>
	Message - returns a pointer to StatusBarTheme.
UWM_GETTBTHEME	<pre>#define UWM_GETTBTHEME (WM_APP + 0x3F09)</pre>
• ····0_ · · · · · · · · · ·	Message - returns a pointer to ToolBarTheme.
UWM_POPUPMENU	#define UWM_POPUPMENU (WM_APP + 0x3F0A)
	Message - creates the menubar popup menu.
UWM_TBRESIZE	#define UWM_TBRESIZE (WM_APP + 0x3F0B)
	Message - sent by toolbar to parent. Used by the rebar.
IIWM TRWINPOSCHANGING	<pre>#define UWM_TBWINPOSCHANGING (WM_APP + 0x3F0C)</pre>
	Message - sent to parent. Toolbar is resizing.
UWM_UPDATECOMMAND	<pre>#define UWM_UPDATECOMMAND (WM_APP + 0x3F0D)</pre>
	Message - sent before a menu is displayed. Used by OnUpdate.
UWM_WINDOWCREATED	<pre>const UINT UWM_WINDOWCREATED = RegisterWindowMe</pre>
	Message - posted when a window is created or attached.
UWN_BARSTART	#define UWN_BARSTART (WM_APP + 0x3F0E)
	Notification - docker bar selected for move.

	Notification - docker bar moved.
UWN_BAREND	<pre>#define UWN_BAREND (WM_APP + 0x3F10) Notification - end of docker bar move.</pre>
UWN_DOCKSTART	<pre>#define UWN_DOCKSTART (WM_APP + 0x3F11) Notification - about to start undocking.</pre>
UWN_DOCKMOVE	<pre>#define UWN_DOCKMOVE (WM_APP + 0x3F12) Notification - undocked docker is being moved.</pre>
UWN_DOCKEND       #define UWN_DOCKEND (WM_APP + 0x3F13)         Notification - docker has been docked.	
UWN_TABCHANGED	<pre>#define UWN_TABCHANGED (WM_APP + 0x3F14) Notification - tab size or position changed.</pre>
UWN_TABDRAGGED <b>#define UWN_TABDRAGGED (WM_APP + 0x3F15)</b> Notification - tab is being dragged.	
UWN_UNDOCKED	<pre>define UWN_UNDOCKED (WM_APP + 0x3F16) Notification - sent by docker when undocked.</pre>

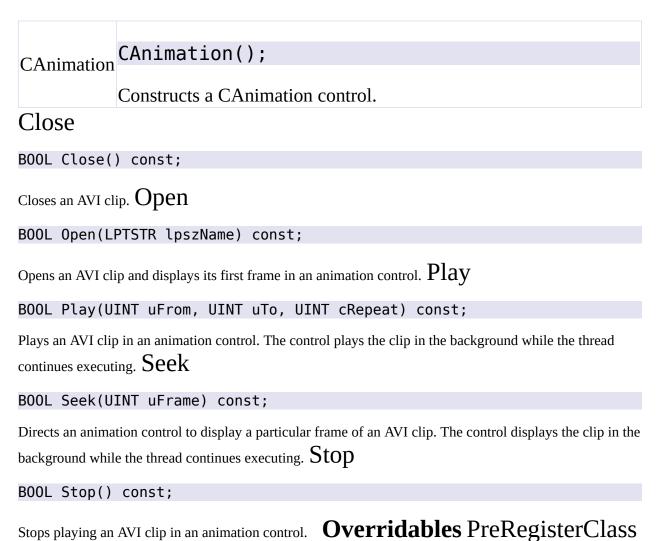
# **CAnimation Class**

### Description

The CAnimation class adds support for the Animation control. An animation control is a rectangular window that displays an AVI (Audio Video Interleaved) format. An AVI clip is a series of bitmap frames, like a movie. Animation controls can only display AVI clips that do not contain audio.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of animation controls.

### **CAnimation Members**



```
virtual void PreRegisterClass(WNDCLASS &wc);
```

Set the window class parameters before the window is created.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

### Remarks

The AVI clip can be added to the application as a resource, or it can be separate file. Only simple AVI files supported by the Windows API animation control can be played.

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CBitmap Class**

## Description

The class responsible for creating and managing bitmap resources.

## **CBitmap Members**

Initialization and Assignment		
CBitmap	CBitmap();	
	CBitmap(HBITMAP hBitmap);	
	CBitmap(LPCTSTR lpstr);	
	CBitmap(int nID);	
	Constructs a CBitmap object.	

#### FromHandle

static CBitmap\* FromHandle(HBITMAP hBitmap);

Returns the CBitmap associated with the specified bitmap handle. If a CBitmap object doesn't already exist, a temporary CBitmap object is created. This temporary CBitmap will be deleted sometime after the

processing of the current message is complete. **Operator HBITMAP()** 

operator HBITMAP() const;

Allows a CBitmap object to be used as a bitmap handle (HBITMAP). Attributes

#### GetBitmapData

BITMAP GetBitmapData() const;

Retrieves the BITMAP structure of the attached bitmap. GetBitmapDimensionEx

CSize GetBitmapDimensionEx() const;

Retrieves the dimensions of a compatible bitmap. The retrieved dimensions must have been set by the

SetBitmapDimensionEx function. GetDIBits

Retrieves the bits of the specified compatible bitmap and copies them into a buffer as a DIB using the specified format. SetBitmapDimensionEx

CSize SetBitmapDimensionEx(int nWidth, int nHeight);

Assigns preferred dimensions to a bitmap. These dimensions can be used by applications; however, they are not used by the system. **SetDIBits** 

Sets the pixels in a compatible bitmap (DDB) using the color data found in the specified DIB. **Operations CreateBitmap** 

HBITMAP CreateBitmap(int nWidth, int nHeight, UINT nPlanes, UINT nBitsPerPixel, LPCVOID lpBits);

Creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel). **CreateBitmapIndirect** 

HBITMAP CreateBitmapIndirect(LPBITMAP lpBitmap);

Creates a bitmap with the width, height, and color format specified in the BITMAP structure. **CreateCompatibleBitmap** 

HBITMAP CreateCompatibleBitmap(CDC\* pDC, int nWidth, int nHeight);

Creates a bitmap compatible with the device that is associated with the specified device context. **CreateDIBitmap** 

HBITMAP CreateDIBitmap(CDC\* pDC, CONST BITMAPINFOHEADER\* lpbmih, DWORD d LPCVOID lpbInit, CONST BITMAPINFO\* lpbmi, UINT uC

Creates a compatible bitmap (DDB) from a DIB and, optionally, sets the bitmap bits. CreateDIBSection

```
HBITMAP CreateDIBSection(CDC* pDC, CONST BITMAPINFO* lpbmi, UINT uColorU
LPVOID* ppvBits, HANDLE hSection, DWORD dwOffse
```

Creates a DIB that applications can write to directly. The function gives you a pointer to the location of the bitmap bit values. CreateMappedBitmap

#### 

Creates a new bitmap using the bitmap data and colors specified by the bitmap resource and the color mapping information. GrayScaleBitmap

#### void GrayScaleBitmap(HBITMAP hbmSource);

Creates a gray scale bitmap from the specified bitmap. LoadBitmap

BOOL LoadBitmap(LPCTSTR lpszName);

B00L LoadBitmap(int nID);

Loads a bitmap from a resource using the resource string. LoadImage

BOOL LoadImage(LPCTSTR lpszName, int cxDesired, int cyDesired, UINT fuLo

BOOL LoadImage(UINT nID, int cxDesired, int cyDesired, UINT fuLoad);

Loads a bitmap from a resource using the resource ID. LoadOEMBitmap

BOOL LoadOEMBitmap(UINT nIDBitmap);

Loads a predefined bitmap. TintBitmap

HIMAGELIST CreateDisabledImageList( HIMAGELIST himlNormal );

Modifies the colour of the supplied Device Dependant Bitmap, by the colour correction values specified. The correction values can range from -255 to +255.

#### **Base class Members**

For base class members, refer to the members of CGDIObject.

#### Remarks

CBitmap objects can be used anywhere a a handle to a bitmap (HBITMAP) might be used. They can be substituted for the HBITMAP in any of the Windows API functions which use a HBITMAP as a function argument. The benefit of using a CBitmap object is that it automatically deletes the bitmap when it is destroyed.

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64	Yes
support	
WinCE support	Yes

# **CBitmapInfoPtr Class**

### Description

The CBitmapInfoPtr class simplifies the creation and use of the BITMAPINFO structure. The BITMAPINFO structure is used in the GetDIBits and SetDIBits Window API functions.

### Remarks

The traditional BITMAPINFO structure is defined like this.

```
typedef struct tagBITMAPINF0 {
  BITMAPINFOHEADER bmiHeader;
  RGBQUAD bmiColors[1];
} BITMAPINF0, *PBITMAPINF0;
```

The thing that makes the traditional BITMAPINFO structure difficult to use is that the bmiColors member is an array. This array has a different number of elements depending on the color depth of the Bitmap. As a result the bmiColors member needs to be created dynamically on the heap before the structure can be declared. CBitmapInfoPtr takes care of these details for for us.

To use CBitmapInfoPtr, simply construct its object by providing a handle (HBITMAP) to the bitmap in its constructor. The CBitmapInfoPtr object can then be used anywhere in place of a pointer to BITMAPINFO. The following example demonstrates the use of CBitmapInfoPtr.

```
// Create our LPBITMAPINFO object
CBitmapInfoPtr pbmi(hbmSource);
```

// Create the DC for GetDIBits to use
CDC MemDC = CreateCompatibleDC(NULL);

// Use GetDIBits to create a DIB from our DDB, and extract the colour da MemDC.GetDIBits(hbmSource, 0, pbmi->bmiHeader.biHeight, NULL, pbmi, DIB\_

Header file	gdi.h
Win32/64	Yes
support	163
WinCE	Yes
support	162

## **CBrush Class**

## Description

The class responsible for creating and managing brush resources.

## **CBrush Members**

Initialization and Assignment	
CBrush	CBrush(); CBrush(HBRUSH hBrush);
	CBrush(COLORREF crColor);
	Constructs a CBrush object.
EromIla	

### FromHandle

static CBrush\* FromHandle(HBRUSH hBrush);

Returns the CBrush associated with the specified brush handle. If a CBrush object doesn't already exist, a temporary CBrush object is created. This temporary CBrush will be deleted sometime after the processing

of the current message is complete. **Operator HBRUSH()** 

```
operator HBRUSH() const;
```

Allows a CBrush object to be used as a brush handle (HBRUSH). Attributes GetLogBrush

LOGBRUSH GetLogBrush() const;

Retrieves the LOGBRUSH structure that defines the style, color, and pattern of a physical brush. **Operations** CreateBrushIndirect

HBRUSH CreateBrushIndirect(LPL0GBRUSH lpLogBrush);

Creates a logical brush from style, color, and pattern specified in the LOGPRUSH struct. **CreateDIBPatternBrush** 

#### HBRUSH CreateDIBPatternBrush(HGLOBAL hglbDIBPacked, UINT fuColorSpec);

 $Creates a logical brush that has the pattern specified by the device-independent bitmap (DIB). \\ CreateDIBPatternBrushPt$ 

HBRUSH CreateDIBPatternBrushPt(LPCVOID lpPackedDIB, UINT nUsage);

Creates a logical brush that has the pattern specified by the device-independent bitmap (DIB). **CreateHatchBrush** 

HBRUSH CreateHatchBrush(int nIndex, COLORREF crColor);

Creates a logical brush that has the specified hatch pattern and color. CreatePatternBrush

HBRUSH CreatePatternBrush(CBitmap\* pBitmap);

Creates a logical brush with the specified bitmap pattern. The bitmap can be a DIB section bitmap, which is created by the CreateDIBSection function, or it can be a device-dependent bitmap.

### CreateSolidBrush

```
HBRUSH CreateSolidBrush(COLORREF crColor);
```

Creates a logical brush that has the specified solid color.

#### **Base class Members**

For base class members, refer to the members of <u>CGDIObject</u>.

#### Remarks

CBrush objects can be used anywhere a a handle to a brush (HBRUSH) might be used. They can be substituted for the HBRUSH in any of the Windows API functions which use a HBRUSH as a function argument. The benefit of using a CBrush object is that it automatically deletes the brush when it is destroyed.

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

# **CButton Class**

## Description

A button is a control the user can click to provide input to an application. The button control provides several types of buttons, including Push Buttons, Check Boxes and Radio Buttons.

CButton is the class responsible for creating a button control. It is typically used in a Dialog, but could also by a child of another window.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of button controls.

## **CButton Members**

#### Construction

CDutton	CButton();
CButton	
	Constructor for CButton.

#### Attributes

GetBitmap	<pre>HBITMAP GetBitmap() const;</pre>
	Returns a handle to the bitmap associated with the button.
CotDuttonStyle	UINT GetButtonStyle() const;
Gelbullolistyle	
	Returns the button style. This can be a combination of possible t
GetCheck	<pre>int GetCheck() const;</pre>
GetCheck	
	Returns the check state of the a radio button or check box.
GetCursor	HCURSOR GetCursor() const;
GeiCuisoi	

	Returns a handle to the cursor associated with the button.
GetIcon	HICON GetIcon() const;
	Returns a handle to the Icon associated with the button.
GetState	UINT GetState() const;
	Returns the state of the button (pushed, checked focused etc).
SetBitmap	<pre>HBITMAP SetBitmap(HBITMAP hBitmap) const;</pre>
SeiDiiniip	Sets the bitmap associated with the button.
SetButtonStyle	<pre>void SetButtonStyle(DWORD dwStyle, BOOL bRed</pre>
SciDuttonotyre	Sets the style of the button.
SetCheck	<pre>void SetCheck(int nCheckState) const;</pre>
betoneek	Sets the check state of the a radio button or check box.
SetCursor	HCURSOR SetCursor(HCURSOR hCursor) const;
	Sets the cursor associated with the button.
SetIcon	HICON SetIcon(HICON hIcon) const;
	Sets the Icon associated with the button.
SetState	<pre>void SetState(B00L bHighlight) const;</pre>
	Sets the state of the button.

#### Overridables

PreRegisterClass	<pre>virtual void PreRegisterClass(WNDCLASS &amp;</pre>	WC
	Set the window class parameters before the window is creat	ed.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

### Remarks

Like all common controls, the button control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a button control.

Refer to the DialogDemo sample to see a demonstration of the CButton class.

Header file	stdcontrols.h	
Win32/64	Yes	
support	163	
WinCE	Yes	
support	165	
Library	Comctl32.lib	
required	Comensz.mo	

## **CClientDC Class**

## Description

The class responsible for creating a device context for the client area of a window.

## **CClientDC Members**

Initialization and Assignment

Initialization and Assignment		
CClientDC	CClientDC(const CWnd* pWnd);	
	Constructs a CClientDC object.	

**Base class Members** 

For base class members, refer to the members of <u>CDC</u>.

## Remarks

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

## **CCmdBar Class**

## Description

The class responsible for creating a Commandbar control. A Commandbar control is supported on the Windows CE operating system. It is not supported on the Win32/64 operating systems.

The command bar is unique to Windows CE. It combines a menu bar, toolbar, and an optional address bar. Windows CE supports multiple command bars, each containing gripper controls that enable users to hide buttons and menus. Command bars can contain combo boxes, edit boxes, and buttons, as well as other types of controls. They also can include the **Close** (**X**) button, the **Help** (**?**) button, and the **OK** button.

CCmdBar	CCmdBar();	
	Constructor for CCmdBar.	
AddAdornments	BOOL AddAdornments(DWORD dwFlags);	
	Adds a button, and optionally, Help and OK buttons, to the corr	
AddBitmap	<pre>int AddBitmap(int idBitmap, int iNumImages</pre>	
	Adds one or more images to the list of button images available	
AddButtons	BOOL AddButtons(int nButtons, TBBUTTON* pT	
	Adds one or more toolbar buttons to a command bar control.	
GetHeight	<pre>int GetHeight() const;</pre>	
	Retrieves the height of the command bar in pixels.	
InsertComboBox	HWND InsertComboBox(int iWidth, UINT dwSty	
	Inserts a combo box into the command bar.	

### **CCmdBar Members**

IsVisible	BOOL IsVisible();
	Retrieves the visibility state of the command bar.
Show	B00L Show(B00L fShow);
	Shows or hides the command bar.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

Header file	wceframe.h
Win32/64	No
support	110
WinCE	Yes
support	105

# **CComboBox Class**

## Description

The CComboBox class is used to create and manage a ComboBox control. A ComboBox control displays a drop-down list of predefined values and an edit field into which the user can enter a value.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of combo box controls.

CComboBox	CComboBox();
	Constructor for the CComboBox.
AddString	<pre>int AddString(LPCTSTR lpszString) con</pre>
	Adds a string to the list.
Clear	<pre>void Clear() const;</pre>
	Deletes the edit selection.
Сору	<pre>void Copy() const;</pre>
	Copies the edit selection to the clipboard.
Cut	<pre>void Cut() const;</pre>
	Deletes the edit selection and places it on the clipboard.
DeleteString	<pre>int DeleteString(int nIndex) const;</pre>
	Deletes a string from the list.

## **CComboBox Members**

Dir	int Dir(UINT attr, LPCTSTR lpszWildCa
	Adds the file names matching the specified attributes an
	<pre>int FindString(int nIndexStart, LPCTS<sup>-</sup></pre>
FindString	Searches the list box of the combo box for an item begir string.
FindStringExact	int FindStringExact(int nIndexStart, ∣
	Returns the index of the first list item exactly matching
GetCount	<pre>int GetCount() const;</pre>
	Returns the number of list items.
GetCurSel	<pre>int GetCurSel() const;</pre>
	Returns the index of the currently selected item, if any.
GetDroppedControlRec	CRect GetDroppedControlRect() const;
GetDioppedControiNec	Fills the specified rectangle structure with the screen co
GetDroppedState	B00L GetDroppedState() const;
GerDroppedState	Returns TRUE if a drop-down list is open, otherwise it r
GetDroppedWidth	<pre>int GetDroppedWidth() const;</pre>
GetDiopped Width	Returns the minimum allowable width, in pixels, of the
GetEditSel	DWORD GetEditSel() const;
	Returns the starting and ending position of the current se
	B00L GetExtendedUI() const;
GetExtendedUI	Returns TRUE if the combo box is a drop-down combo

	user-interface flag is set, otherwise it returns FALSE.
GetHorizontalExtent	<pre>int GetHorizontalExtent() const; Returns the scrollable width, in pixels, of the drop-dowr</pre>
GetItemData	DWORD GetItemData(int nIndex) const; Returns the value associated with the specified list item.
GetItemHeight	<pre>int GetItemHeight(int nIndex) const; Returns the height, in pixels, of the specified owner-dray</pre>
GetLBText	<pre>int GetLBText(int nIndex, LPTSTR lpsz Copies the specified list text to the specified buffer.</pre>
GetLBTextLen	<pre>int GetLBTextLen(int nIndex) const; Returns the length, in TCHARs, of the specified list text</pre>
GetLocale	LCID GetLocale() const; Returns the current locale for the list.
GetTopIndex	<pre>int GetTopIndex() const; Returns the index of the first visible item in the drop-do</pre>
InitStorage	int InitStorage(int nItems, int nBytes Initializes space for the specified number of items and th strings.
InsertString	<pre>int InsertString(int nIndex, LPCTSTR Inserts a list item at the specified position.</pre>

LimitText	<pre>void LimitText(int nMaxChars) const;</pre>
	Sets the maximum number of characters a user can ente
Paste	<pre>void Paste() const;</pre>
	Replaces the edit selection with the contents of the clipb
ResetContent	<pre>void ResetContent() const;</pre>
	Removes the contents of the list.
SelectString	<pre>int SelectString(int nStartAfter, LPC</pre>
	Selects the first list item, if any, that begins with the cha
SetCurSel	<pre>int SetCurSel(int nIndex) const;</pre>
	Sets the current selection.
SetDroppedWidth	<pre>int SetDroppedWidth(int nWidth) const</pre>
1 Г	Sets the minimum allowable width, in pixels, of the droj
SetEditSel	BOOL SetEditSel(int nStartChar, int n
	Selects the specified range of text.
	<pre>int SetExtendedUI(B00L bExtended = TR</pre>
SetExtendedUI	Sets or clears the extended user-interface flag. This flag the list in a drop-down list.
SetHorizontalExtent	<pre>void SetHorizontalExtent(UINT nExtent</pre>
	Sets the scrollable width, in pixels, of the drop-down lis
SetItemData	int SetItemData(int nIndex, DWORD dwI <sup>.</sup>
	Associates the specified value with a list item.

SetItemHeight	<pre>int SetItemHeight(int nIndex, UINT cy: Sets the height of the specified owner-drawn list item or</pre>
SetLocale	LCID SetLocale( LCID NewLocale ) cons <sup>-</sup> Sets the current local for the list.
SetTopIndex	SetTopIndex(int nIndex) const; Scroll the drop-down list so the specified item is at the te
ShowDropDown	<pre>void ShowDropDown(BOOL bShow = TRUE) Shows or hide the drop-down list.</pre>

#### Overridables

PreRegisterClass	virtual void PreRegisterClass(WNDCLASS &wo	С
	Set the window class parameters before the window is created	

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

### Remarks

Like all common controls, the combobox control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for an combobox control.

Header file	controls.h
Win32/64	Yes
support	163
WinCE	Yes
support	105

# **CComboBoxEx Class**

## Description

The CComboBoxEx class is used to create and manage a ComboBoxEx control. ComboBoxEx controls are an extension of the combo box control that provides native support for item images.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of ComboBoxEx controls.

CComboBoxEx	CComboBoxEx(); Constructor for the ComboBoxEx.	
DeleteItem	<pre>int DeleteItem(int nIndex ) const; Removes an item.</pre>	
GetComboBoxCtrl	HWND GetComboBoxCtrl() const; Retrieves the handle to the child combo box control.	
GetEditCtrl	HWND GetEditCtrl() const; Retrieves the handle to the edit control portion of a ComboBc	
GetExtendedStyle	DWORD GetExtendedStyle() const; Retrieves the extended style that are in use for a ComboBoxE	
GetImageList	CImageList* GetImageList() const; Retrieves the handle to the image list assigned to the ComboE	

## **CComboBoxEx Members**

GetItem	BOOL GetItem(COMBOBOXEXITEM* pCBItem) con
	Retrieves item information for a given ComboBoxEx item.
HasEditChanged	B00L HasEditChanged () const;
	Returns TRUE if the user has changed text of the ComboBox
InsertItem	<pre>int InsertItem(COMBOBOXEXITEM* lpcCBItem)</pre>
	Inserts a new item in the ComboBoxEx control.
SetExtendedStyle	DWORD SetExtendedStyle(DWORD dwExMask, DW
	Sets the extend styles for the ComboBoxEx controls.
SetImageList	CImageList* SetImageList(CImageList* pNew
	Sets the image list for the ComboBoxEx control.
SetItem	BOOL SetItem(PCOMBOBOXEXITEM lpcCBItem) c
	Sets the attributes for an item of the ComboBoxEx control.

#### Overridables

PreRegisterClass	virtual void PreRegisterClass(WNDCLASS &w	/C
	Set the window class parameters before the window is created	d.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Like all common controls, the ComboBoxEx control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a ComboBoxEx control.

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes

# **CCriticalSection Class**

## Description

Critical sections are a feature of the Windows API. They are used to prevent a section of code from being executed by more than one thread at a time. Once a thread has locked a critical section, any other thread wishing to execute that code must wait until the critical section is released so it can lock the critical section.

For example, we might wish to prevent a situation where one thread is changing a global variable while another thread is using it. The CCriticalSection class provides a convenient and object orientated approach to using critical sections. It is used internally by the Win32++ library, but will also be useful to developers when coding their own multi-threaded applications.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of critical sections to assist with synchronisation in multi-threaded applications.

## **CCriticalSection Members**

CCriticalSection	CCriticalSection(); Constructor for CCriticalSection.
Lock	<pre>void Lock(); Enter a critical section. Only one thread at a time runs code protected by a critical section.</pre>
Release	void Release(); Leave a critical section.

## Remarks

The following example demonstrates the use of CCriticalSection.

```
// m_csVar is a class member variable of type CCriticalSection
// Lock the critical section while we change its value
```

```
m_csVar.Lock();
// Modify the global variable
g_Var = 10;
// Release the critical section
m_csVar.Release();
// Note: we would also protect any reads from the global
// variable with the same m_csVar CCriticalSection
```

Header file	wincore.h
Win32/64	Yes
support	103
WinCE	Yes
support	165

# **CDateTime Class**

## Description

The CDateTime class adds support for the date and time picker control. A date and time picker (DTP) control provides a simple and intuitive interface through which to exchange date and time information with a user.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of date and time picker controls.

## **CDateTime Members**

## **Construction** CDateTime

CDateTime();

Constructs the date and time picker control. Attributes GetMonthCalColor

COLORREF GetMonthCalColor(int iColor) const;

Gets the color for a given portion of the month calendar within a date and time picker (DTP) control. GetMonthCalCtrl

```
HWND GetMonthCalCtrl() const;
```

Gets the handle to a date and time picker's (DTP) child month calendar control. GetMonthCalFont

CFont\* GetMonthCalFont() const;

Gets the font that the date and time picker (DTP) control's child month calendar control is currently using. GetRange

DWORD GetRange(LPSYSTEMTIME lpSysTimeArray) const;

Gets the current minimum and maximum allowable system times for the date and time picker (DTP) control. GetTime

DWORD GetTime(LPSYSTEMTIME pTimeDest) const;

Gets the currently selected time from a date and time picker (DTP) control and places it in a specified SYSTEMTIME structure. SetMonthCalColor

COLORREF SetMonthCalColor(int iColor, COLORREF ref);

Sets the color for a given portion of the month calendar within a date and time picker (DTP) control. **SetFormat** 

B00L SetFormat(LPCTSTR pstrFormat);

Sets the display of a date and time picker (DTP) control based on a given format string.

## SetMonthCalFont

void SetMonthCalFont(HFONT hFont, BOOL bRedraw = TRUE);

Sets the font to be used by the date and time picker (DTP) control's child month calendar control. **SetRange** 

#### SetKange

BOOL SetRange(DWORD flags, LPSYSTEMTIME lpSysTimeArray);

Sets the minimum and maximum allowable system times for the date and time picker (DTP) control. SetTime

BOOL SetTime(DWORD flag, LPSYSTEMTIME pTimeNew = NULL);

Sets the date and time picker (DTP) control to a given date and time. **Overridables** 

## PreRegisterClass

virtual void PreRegisterClass(WNDCLASS &wc);

Sets a date and time picker (DTP) control to a given date and time.

**Base class Members** 

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Date and time picker (DTP) controls have several Date and Time Picker Control Styles that determine a control's appearance and behavior. Specify the style when creating the control with the *dwStyle* parameter of CreateWindowEx. To retrieve or change the window style after you have created the control, use GetWindowLongPtr and SetWindowLongPtr.

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CDC Class**

## Description

A device context is a Windows data structure containing information about the drawing attributes of a device such as a display or a printer. All drawing calls are made through a device-context object, which encapsulates the Windows APIs for drawing lines, shapes, and text. Device contexts allow deviceindependent drawing in Windows.

The CDC class provides a GDI device context, along with a set of member functions to perform the various tasks with the device context. Member functions can also be used to create the various GDI objects, such as brushes and pens. Alternatively, GDI objects can be attached to the device context using SelectObject.

## **CDC Members**

## **Initialisation and Assignment** CDC CDC(); CDC(HDC hDC, HWND hWnd = 0);

Constructor for CDC. Attach

void Attach(HDC hDC);

Attach an existing device context to the CDC object. Detach

#### HDC Detach();

Detach a device context from the CDC object. The device context will be destroyed when the CDC is destroyed, unless it is detached. FromHandle

static CDC\* FromHandle(HDC hDC);

Returns the CDC associated with the specified device context handle. If a CDC object doesn't already exist, a temporary CDC object is created. This temporary CDC will be deleted sometime after the processing of the current message is complete. GetHDC

HDC GetHDC() const;

Returns the Device Context handle associated with this CDC object. RestoreDC

B00L RestoreDC(int nSavedDC) const;

Restores the DC to the specified state. The state includes selected objects and mapping modes. SaveDC

int SaveDC() const;

Saves the state of the device context. **Operator HDC ()** 

operator HDC() const;

Allow a CDC object to be used as a handle to a device context (HDC). Operator HDC =

```
void operator = (const HDC hDC);
```

Used to assign a HDC to a CDC when it is created. Create and Select Bitmaps

## CreateBitmap

void CreateBitmap(int cx, int cy, UINT Planes, UINT BitsPerPixel, LPCVOI

Creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel). **CreateBitmapIndirect** 

void CreateBitmapIndirect(LPBITMAP pBitmap);

Creates a bitmap with the specified width, height, and color format (color planes and bits-per-pixel), specified in the BITMAP struct. CreateCompatibleBitmap

void CreateCompatibleBitmap(CDC\* pDC, int cx, int cy);

Creates a bitmap compatible with the device that is associated with the specified device context. **CreateDIBSection** 

Creates a DIB that applications can write to directly. The function gives you a pointer to the location of the bitmap bit values. You can supply a handle to a file-mapping object that the function will use to create the

bitmap, or you can let the system allocate the memory for the bitmap. CreateDIBitmap

void CreateDIBitmap(CDC\* pDC, const BITMAPINFOHEADER& bmih, DWORD fdwInit, LPCVOID lpbInit, BITMAPINFO& bmi, UINT fuUsage

Creates a compatible bitmap (DDB) from a DIB and, optionally, sets the bitmap bits.

## CreateMappedBitmap

#### void CreateMappedBitmap(UINT nIDBitmap, UINT nFlags /\*= 0\*/, LPCOLORMAP lpColorMap /\*= NULL\*/, int nMapSize /\*

Creates a bitmap for use in a toolbar. DetachBitmap

CBitmap DetachBitmap();

Provides a convenient method of detaching a bitmap from a memory device context. GetBitmapData

BITMAP GetBitmapData() const;

Retrieves the current bitmap information. GetCurrentBitmap

CBitmap\* GetCurrentBitmap() const;

Retrieves a pointer to the currently selected bitmap. LoadBitmap

B00L LoadBitmap(UINT nID);

BOOL LoadBitmap(LPCTSTR lpszName);

Loads the specified bitmap resource from a module's executable file. LoadImage

BOOL LoadImage(UINT nID, int cxDesired, int cyDesired, UINT fuLoad);

BOOL LoadImage(LPCTSTR lpszName, int cxDesired, int cyDesired, UINT fuLo

Loads the specified bitmap resource from a module's executable file. LoadOEMBitmap

BOOL LoadOEMBitmap(UINT nIDBitmap);

Loads a predefined bitmap used by Windows. SelectObject

CBitmap\* SelectObject(const CBitmap\* pBitmap);

## Attach an existing bitmap to the CDC. **Create and Select Brushes CreateBrushIndirect**

void CreateBrushIndirect(LPLOGBRUSH pLogBrush);

Creates a logical brush that has the specified style, color, and pattern. **CreateDIBPatternBrush** 

void CreatePatternBrush(HBITMAP hbmp);

Creates a logical brush that has the pattern specified by the specified device-independent bitmap (DIB).

## CreateDIBPatternBrushPt

#### void CreateDIBPatternBrushPt(LPCV0ID lpPackedDIB, UINT iUsage);

 $Creates a logical brush that has the pattern specified by the device-independent bitmap (DIB). \\ CreateHatchBrush$ 

void CreateHatchBrush(int fnStyle, COLORREF rgb);

Creates a logical brush that has the specified hatch pattern and color. CreatePatternBrush

void CreatePatternBrush(CBitmap\* pBitmap);

Creates a logical brush with the specified bitmap pattern. CreateSolidBrush

void CreateSolidBrush(COLORREF rbg);

Creates a logical brush that has the specified solid color. GetCurrentBrush

CBrush\* GetCurrentBrush() const;

Retrieves a pointer to the currently selected brush. GetLogBrush

LOGBRUSH GetLogBrush() const;

Retrieves the current brush information. SelectObject

CBrush\* SelectObject(const CBrush\* pBrush);

Attach an existing brush to the CDC. Create and Select Fonts CreateFont

void CreateFont(int nHeight, int nWidth, int nEscapement, int nOrientati DWORD fdwItalic, DWORD fdwUnderline, DWORD fdwStrikeOut, DWORD DWORD fdwOutputPrecision, DWORD fdwClipPrecision, DWORD fdwQua DWORD fdwPitchAndFamily, LPCTSTR lpszFace);

Creates a logical font with the specified characteristics. CreateFontIndirect

#### void CreateFontIndirect(LPL0GF0NT plf);

Creates a logical font that has the specified characteristics. GetCurrentFont

CFont\* GetCurrentFont() const;

Retrieves a pointer to the currently selected font. GetLogFont

LOGFONT GetLogFont() const;

Retrieves the current font information. SelectObject

CFont\* SelectObject(const CFont\* pFont);

Attach an existing font to the CDC. **Create and Select Palettes** CreateHalftonePalette

void CreateHalftonePalette();

Creates and selects a half tone palette into the CDC. CreatePalette

void CreatePalette(LPLOGPALETTE pLogPalette);

Creates and selects a paletted into the CDC. SelectPalette

CPalette\* SelectPalette(const CPalette\* pPalette, BOOL bForceBkgnd);

Attach an existing palette to the CDC. Create and Select Pens CreatePen

void CreatePen(int nStyle, int nWidth, COLORREF rgb);

Creates a logical pen that has the specified style, width, and color. CreatePenIndirect

void CreatePenIndirect(LPL0GPEN pLogPen);

Creates a logical pen that has the style, width, and color specified in a structure. GetCurrentPen

CPen\* GetCurrentPen() const;

Retrieves a pointer to the currently selected pen. GetLogPen

LOGPEN GetLogPen();

Retrieves the current pen information. SelectObject

CPen\* SelectObject(const CPen\* pPen);

Attach an existing pen to the CDC. **Retrieve and Select Stock Objects** GetStockObject

HGDIOBJ GetStockObject(int nIndex) const;

Retrieves a stock brush, pen, or font into the device context.

Possible nIndex values: BLACK\_BRUSH, DKGRAY\_BRUSH, DC\_BRUSH, HOLLOW\_BRUSH, LTGRAY\_BRUSH, NULL\_BRUSH, WHITE\_BRUSH, BLACK\_PEN, DC\_PEN, ANSI\_FIXED\_FONT, ANSI\_VAR\_FONT, DEVICE\_DEFAULT\_FONT, DEFAULT\_GUI\_FONT, OEM\_FIXED\_FONT,

SYSTEM\_FONT, or SYSTEM\_FIXED\_FONT. SelectStockObject

HGDI0BJ SelectStock0bject(int nIndex);

Selects a stock brush, pen, or font into the device context. Possible nIndex values: BLACK\_BRUSH, DKGRAY\_BRUSH, DC\_BRUSH, HOLLOW\_BRUSH, LTGRAY\_BRUSH, NULL\_BRUSH, WHITE\_BRUSH, BLACK\_PEN, DC\_PEN, ANSI\_FIXED\_FONT, ANSI\_VAR\_FONT, DEVICE\_DEFAULT\_FONT, DEFAULT\_GUI\_FONT, OEM\_FIXED\_FONT,

## SYSTEM\_FONT, or SYSTEM\_FIXED\_FONT. **Create Regions** CreateEllipticRgn

int CreateEllipticRgn(int left, int top, int right, int bottom);

Creates an elliptical region from the specified rectangle co-ordinates.

## CreateEllipticRgnIndirect

int CreateEllipticRgnIndirect(const RECT& rc);

Creates an elliptical region from the specified RECT. CreatePolygonRgn

int CreatePolygonRgn(LPPOINT ppt, int cPoints, int fnPolyFillMode);

Creates a polygonal region from an array of points. CreatePolyPolygonRgn

Creates the polygon region from a series of polygons. The polygons can overlap. CreateRectRgn

int CreateRectRgn(int left, int top, int right, int bottom);

Creates a rectangular region from the specified rectangle co-ordinates.

## CreateRectRgnIndirect

int CreateRectRgnIndirect(const RECT& rc);

Creates a rectangular region from a specified RECT.

## Wrappers for Window API functions

Initialization

CreateCompatibleDC	BOOL CreateCompatibleDC(CDC* pDC);
	Creates a memory device context (DC) compatible with the specified device.
CreateDC	BOOL CreateDC(LPCTSTR lpszDriver, LPCTSTR lpszDevice, LPCTSTR lpszOutput, const DEVMODE* pInitData);
	Creates a device context (DC) for a device using the specified name.
CreateIC	BOOL CreateIC(LPCTSTR lpszDriver, LPCTSTR lpszDevice, LPCTSTR lpszOutput, const DEVMODE* pInitData);
	Creates an information context for the specified device. The information context prodevice without creating a device context (DC).
EnumObjects	<pre>int EnumObjects(int nObjectType, GOBJENUMPROC lpObjectFu</pre>
	Enumerates the pens or brushes available for the device context. This function calls each available object, supplying data describing that object.
GetDeviceCaps	<pre>int GetDeviceCaps(int nIndex) const;</pre>
	Retrieves device-specific information for the specified device.

#### Point and Line Drawing Functions

AngleArc	BOOL AngleArc(int x, int y, int nRadius, float fStartAng float fSweepAngle) const;
	Draws a line segment and an arc. The line segment is drawn from the current positic along the perimeter of a circle with the given radius and center. The length of the ar
	BOOL Arc(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) const;
Arc	
	BOOL Arc(RECT& rc, POINT ptStart, POINT ptEnd) const;
	Draws an elliptical arc.
	BOOL ArcTo(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) const;
ArcTo	
	BOOL ArcTo(RECT& rc, POINT ptStart, POINT ptEnd) const;
	Draws an elliptical arc.
CloseFigure	
	BOOL CloseFigure() const;
	Closes the figure by drawing a line from the current position to the first point of the

GetArcDirection	<pre>int GetArcDirection() const;</pre>
	Retrieves the current arc direction for the specified device context. Arc and rectain
GetCurrentPositition	<pre>CPoint GetCurrentPosition() const;</pre>
	Returns the current drawing position.
GetMiterLimit	BOOL GetMiterLimit(PFLOAT peLimit) const;
	Retrieves the miter limit for the device context. The miter limit is used when drav
	COLORREF GetPixel(int x, int y) const;
GetPixel	COLORREF GetPixel(POINT pt) const;
	Retrieves the red, green, blue (RGB) color value of the pixel at the specified coor
	<pre>int GetR0P2() const;</pre>
GetROP2	Retrieves the foreground mix mode of the specified device context. The mix mode already on the screen are combined to yield a new color.
	BOOL LineTo(int x, int y) const;
LineTo	BOOL LineTo(POINT pt) const;
	Draws a line from the current position up to, but not including, the specified poin
	CPoint MoveTo(int x, int y) const;
MoveTo	CPoint MoveTo(POINT pt) const;
	Updates the current position to the specified point.
PolyBezier	BOOL PolyBezier(const POINT* lpPoints, int nCount) con
	Draws one or more Bezier curves.
PolyBezierTo	B00L PolyBezierTo(const P0INT* lpPoints, int nCount) c
r ory Dezler 10	Draws one or more Bezier curves.
PolyDraw	B00L PolyDraw(const P0INT* lpPoints, const BYTE* lpTyp
l oly Diaw	Draws a set of line segments and Bezier curves.
Polyline	BOOL Polyline(LPPOINT lpPoints, int nCount) const;
rorynne	Draws a series of line segments by connecting the points in the specified array.
PolylineTo	BOOL PolylineTo(const POINT* lpPoints, int nCount) con

	Draws one or more straight lines.
PolyPolyline	BOOL PolyPolyline(const POINT* lpPoints, const DWORD* lp int nCount) const;
	Draws multiple series of connected line segments.
SetArcDirection	<pre>int SetArcDirection(int nArcDirection) const;</pre>
	Sets the drawing direction to be used for arc and rectangle functions.
SetMiterLimit	BOOL SetMiterLimit(FLOAT eNewLimit, PFLOAT peOldLimit) c
	Sets the miter limit for the device context. The miter limit is used when drawing geo
	COLORREF SetPixel(int x, int y, COLORREF crColor) const;
SetPixel	COLORREF SetPixel(POINT pt, COLORREF crColor) const;
	Sets the pixel at the specified coordinates to the specified color.
SetPixelV	BOOL SetPixelV(int x, int y, COLORREF crColor) const;
	BOOL SetPixelV(POINT pt, COLORREF crColor) const;
	Sets the pixel at the specified coordinates to the closest approximation of the specifi
SetROP2	<pre>int SetROP2(int iDrawMode) const;</pre>
	Sets the current foreground mix mode. GDI uses the foreground mix mode to combi colors already on the screen.

## Shape Drawing Functions

	BOOL Chord(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) const;
Chord	BOOL Chord(const RECT& rc, POINT ptStart, POINT ptEnd) c
	Draws a chord (a region bounded by the intersection of an ellipse and a line segmen the current pen and filled by using the current brush
DrawFocusRect	<pre>void DrawFocusRect(const RECT&amp; rc) const;</pre>
Didwrocuskeci	Draws a rectangle in the style used to indicate that the rectangle has the focus.
Ellipse	BOOL Ellipse( int x1, int y1, int x2, int y2 ) const;
	BOOL Ellipse(const RECT& rc) const;
	Draws an ellipse. The center of the ellipse is the center of the specified bounding reapen and is filled by using the current brush.

	BOOL Pie(int x1, int y1, int x2, int y2, int x3, int y3,
Pie	BOOL Pie(const RECT& rc, POINT ptStart, POINT ptEnd) cor
	Draws a pie-shaped wedge bounded by the intersection of an ellipse and two radials filled by using the current brush.
	BOOL Polygon(LPPOINT lpPoints, int nCount) const;
Polygon	Draws a polygon consisting of two or more vertices connected by straight lines. The filled by using the current brush and polygon fill mode.
	BOOL PolyPolygon(LPPOINT lpPoints, LPINT lpPolyCounts, i
PolyPolygon	Draws a series of closed polygons. Each polygon is outlined by using the current pe fill mode. The polygons drawn by this function can overlap.
	BOOL Rectangle(int x1, int y1, int x2, int y2) const;
Rectangle	BOOL Rectangle( const RECT& rc ) const;
	Draws a rectangle. The rectangle is outlined by using the current pen and filled by u
	BOOL RoundRect(int x1, int y1, int x2, int y2, int nWidt
RoundRect	BOOL RoundRect(const RECT& rc, int nWidth, int nHeight)
	Draws a rectangle with rounded corners. The rectangle is outlined by using the curre
Fill and Image Draw	ing functions
DrawEdge	BOOL DrawEdge(const RECT& rc, UINT nEdge, UINT nFlags) c
	Draws one or more edges of the specified rectangle.
DrawFrameControl	BOOL DrawFrameControl(const RECT& rc, UINT nType, UINT r
	Draws a frame control of the specified type and style.
DrawIcon	BOOL DrawIcon(int x, int y, HICON hIcon) const;
	BOOL DrawIcon(POINT point, HICON hIcon) const;
	Draws an icon or cursor into the specified device context.
DrawIconEx	BOOL DrawIconEx(int xLeft, int yTop, HICON hIcon, int cx INT istepIfAniCur, CBrush* pFlickerFreeDraw
	Draws an icon or cursor into the specified device context, performing the specified a icon or cursor as specified.

FillRect	BOOL FillRect(const RECT& rc, CBrush* pBrushr) const;
	Fills a rectangle by using the specified brush. This function includes the left and top of the rectangle.
FillRgn	BOOL FillRgn(CRgn* pRgn, CBrush* pBrush) const;
	Fills a region by using the specified brush.
	BOOL FrameRect(const RECT& rc, CBrush* pBrush) const;
FrameRect	Draws a border around the specified rectangle by using the specified brush. The wic unit.
FrameRgn	BOOL FrameRgn(CRgn* pRgn, CBrush* pBrush, int nWidth, ir
	Draws a border around the specified region by using the specified brush.
GetPolyFillMode	<pre>int GetPolyFillMode() const;</pre>
Ged off introde	Retrieves the current polygon fill mode.
GradientFill	<pre>void GradientFill(COLORREF Color1, COLORREF Color2, cons</pre>
	Fills the specified rectangle with a color gradient.
InvertRect	BOOL InvertRect(const RECT& rc) const;
	Inverts a rectangle in a window by performing a logical NOT operation on the color
PaintRgn	BOOL PaintRgn(CRgn* pRgn) const;
i uniti gii	Paints the specified region by using the brush currently selected into the device cont
SetPolyFillMode	<pre>int SetPolyFillMode(int iPolyFillMode) const;</pre>
See of yr minode	Sets the polygon fill mode for functions that fill polygons.
SolidFill	<pre>void SolidFill(COLORREF Color, const RECT&amp; rc) const;</pre>
Sonarm	Fills a rectangle with a solid color.
Bitmap Functions	
BitBlt	BOOL BitBlt(int x, int y, int nWidth, int nHeight, CDC* int xSrc, int ySrc,
	Performs a bit-block transfer of the color data corresponding to a rectangle of pixels
	<pre>void DrawBitmap(int x, int y, int cx, int cy, HBITMAP ht</pre>
DrawBitmap	Draws the specified bitmap to the specified DC using the mask colour provided as t

	Window DC or a memory DC.	
ExtFloodFill	BOOL ExtFloodFill(int x, int y, COLORREF crColor, UIN	
	Fills an area of the display surface with the current brush.	
FloodFill	BOOL FloodFill(int x, int y, COLORREF crColor) const;	
rioourin	Fills an area of the display surface with the current brush. The area is assumed	
GetDIBits	int GetDIBits(CBitmap* pBitmap, UINT uStartScan, UINT LPVOID lpvBits, LPBITMAPINFO lpbi, UINT	
	Retrieves the bits of the specified compatible bitmap and copies them into a buff	
	<pre>int GetStretchBltMode() const;</pre>	
GetStretchBltMode	Retrieves the current stretching mode. The stretching mode defines how color da stretched or compressed when the StretchBlt function is called.	
	BOOL MaskBlt(int nXDest, int nYDest, int nWidth, int n int nXSrc, int nYSrc, CBitmap* pMask, in DWORD dwRop) const;	
MaskBlt	Combines the color data for the source and destination bitmaps using the specifi Parameters: nXDest x-coord of destination upper-left corner nYDest y-coord of destination upper-left corner nWidth width of source and destination nHeight height of source and destination pSrc pointer to source DC nXSrc x-coord of upper-left corner of source nYSrc y-coord of upper-left corner of source pMask pointer to monochrome bit mask xMask horizontal offset into mask bitmap yMask vertical offset into mask bitmap dwRop raster operation code	
PatBlt	BOOL PatBlt(int x, int y, int nWidth, int nHeight, DWOR Paints the specified rectangle using the brush that is currently selected into the devicolors are combined by using the specified raster operation.	
SetDIBits	int SetDIBits(CBitmap* pBitmap, UINT uStartScan, UINT CONST VOID *lpvBits, LPBITMAPINFO lpbi,	
	Sets the pixels in a compatible bitmap (DDB) using the color data found in the s	
SetStretchBltMode	<pre>int SetStretchBltMode(int iStretchMode) const;</pre>	
Setonetemphillioue	Sets the bitmap stretching mode in the device context.	

	BOOL StretchBlt(int x, int y, int nWidth, int nHeight, ( int ySrc, int nSrcWidth, int nSrcHeight,
	Copies a bitmap from a source rectangle into a destination rectangle, stretching or codestination rectangle, if necessary. Parameters:
StretchBlt	xx-coord of destination upper-left corneryy-coord of destination upper-left cornernWidthwidth of destination rectanglenHeightheight of destination rectanglepSrcDChandle to source DCxSrcx-coord of source upper-left cornerySrcy-coord of source upper-left cornernSrcWidthwidth of source rectanglenSrcHeightheight of source rectangledwRopraster operation code
StretchDIBits	int StretchDIBits(int XDest, int YDest, int nDestWidth, int YSrc, int nSrcWidth, int nSrcHeigh BITMAPINFO& bi, UINT iUsage, DWORD dwF
	Copies the color data for a rectangle of pixels in a DIB to the specified destination r the source rectangle, this function stretches the rows and columns of color data to fi rectangle is smaller than the source rectangle, this function compresses the rows and
	BOOL TransparentBlt(int x, int y, int nWidth, int hHeigh int xSrc, int ySrc, int nWidthSrc, UINT crTransparent) const;
TransparentBlt	Performs a bit-block transfer of the color data corresponding to a rectangle of pixels destination device context. Parameters: x x-coord of destination upper-left corner y y-coord of destination upper-left corner nWidth width of destination rectangle hHeight height of destination rectangle pSrcDC pointer to source DC xSrc x-coord of source upper-left corner ySrc y-coord of source upper-left corner nWidthSrc width of source rectangle nHeightSrc height of source rectangle crTransparent color to make transparent
Palette and color Fun	ctions
GetColorAdjustment	BOOL GetColorAdjustment(LPCOLORADJUSTMENT pCA) const; Retrieves the color adjustment values for the device context.
GetCurrentPalette	<pre>CPalette* GetCurrentPalette() const;</pre>
	Retrieves a pointer to the currently selected palette

GetNearestColor	COLORREF GetNearestColor(COLORREF crColor) const; Retrieves a color value identifying a color from the system palette that will be displa	
RealizePalette	<pre>void RealizePalette() const;</pre>	
	Use this to realize changes to the device context palette.	
SetColorAdjustment	B00L SetColorAdjustment(CONST COLORADJUSTMENT* pCA) cons	
	Sets the color adjustment values for the device context.	
UpdateColors	BOOL UpdateColors() const;	
opulatecolors	Updates the client area of the specified device context by remapping the current collogical palette.	
Clipping and Region	Functions	
	BOOL BeginPath() const;	
BeginPath	Opens a path bracket in the device context.	
EndPath	B00L EndPath() const;	
	Combines the specified region with the current clipping region using the specified n	
	int ExcludeClipRect(int Left, int Top, int Right, int Bc	
ExcludeClipRect	<pre>int ExcludeClipRect(const RECT&amp; rc) const;</pre>	
	Creates a new clipping region that consists of the existing clipping region minus the	
ExtSelectClipRgn	<pre>int ExtSelectClipRgn(CRgn* pRgn, int fnMode) const;</pre>	
	Combines the specified region with the current clipping region using the specified n	
FlattenPath	BOOL FlattenPath() const;	
	Transforms any curves in the path that is selected into the device context, turning ea	
GetClipBox	<pre>int GetClipBox(RECT&amp; rc) const;</pre>	
CetempBon	Retrieves the dimensions of the tightest bounding rectangle that can be drawn aroun area is defined by the current clipping region or clip path, as well as any overlapping	
	int GetPath(POINT* pPoints, BYTE* pTypes, int nCount) cc	
GetClipPath	Retrieves the coordinates defining the endpoints of lines and the control points of cu device context. pPoints: An array of POINT structures that receives the line endpoints and curve co	

	pTypes: Pointer to an array of bytes that receives the vertex types (PT_MOVET nCount: The total number of POINT structures that can be stored in the array p
GetClipRgn	<pre>int GetClipRgn(HRGN hrgn) const;</pre>
	Retrieves a handle identifying the current application-defined clipping region for
	<pre>int IntersectClipRect(int Left, int Top, int Right, i</pre>
IntersectClipRect	<pre>int IntersectClipRect(const RECT&amp; rc) const;</pre>
	Creates a new clipping region from the intersection of the current clipping region
OffsetClipRgn	<pre>int OffsetClipRgn(int nXOffset, int nYOffset) const;</pre>
	Moves the clipping region of the device context by the specified offsets.
PtVisible	BOOL PtVisible(int X, int Y) const;
	Determines whether the specified point is within the clipping region of a device
RectVisible	BOOL RectVisible(const RECT& rc) const;
	Determines whether any part of the specified rectangle lies within the clipping
	BOOL SelectClipPath(int nMode) const;
SelectClipPath	Selects the current path as a clipping region for the device context, combining t the specified mode.
SelectClipRgn	<pre>int SelectClipRgn(CRgn* pRgn) const;</pre>
	Selects a region as the current clipping region for the device context.
WidenDath	BOOL WidenPath() const;
WidenPath	Redefines the current path as the area that would be painted if the path were structure context.
Co-ordinate Funct	ions
	BOOL DPtoLP(LPPOINT lpPoints, int nCount) const;
DPtoLP	BOOL DPtoLP(LPRECT lpRect) const;
	Converts device coordinates into logical coordinates. The conversion depends of the origins and extents for the window and viewport, and the world transform
	BOOL LPtoDP(LPPOINT lpPoints, int nCount) const;
LPtoDP	BOOL LPtoDP(LPRECT lpRect) const;

	Converts logical coordinates into device coordinates. The conversion depends of of the origins and extents for the window and viewport, and the world transform
Layout Functions	
GetLayout	DWORD GetLayout() const;
	Returns the layout of a device context (DC). Can be used to retrieve the LAYOU
SetLayout	DWORD SetLayout(DWORD dwLayout) const;
Seillayout	Changes the layout of a device context (DC). Can be used to set the LAYOUT_
Mapping Functions	
GetMapMode	<pre>int GetMapMode() const;</pre>
Gettiniphilode	Retrieves the current mapping mode.
GetViewportExtEx	BOOL GetViewportExtEx(LPSIZE lpSize) const;
	Retrieves the x-extent and y-extent of the current viewport for the specified devi
GetViewportOrgEx	BOOL GetViewportOrgEx(LPPOINT lpPoint) const;
	Retrieves the x-coordinates and y-coordinates of the window origin for the spec
GetWindowExtEx	BOOL GetWindowExtEx(LPSIZE lpSize) const;
	Retrieves the x-extent and y-extent of the window for the specified device conte
GetWindowOrgEx	BOOL GetWindowOrgEx(LPPOINT lpPoint) const;
	Retrieves the x-coordinates and y-coordinates of the window origin for the spec
OffsetViewportOrgEx	B00L OffsetViewportOrgEx(int nWidth, int nHeight, LPP
	Modifies the viewport origin for a device context using the specified horizontal
OffsetWindowOrg	BOOL OffsetWindowOrg(int nWidth, int nHeight, LPPOINT
	Modifies the viewport for a device context using the ratios formed by the specifi
OffsetWindowOrgEx	BOOL OffsetWindowOrgEx(int nWidth, int nHeight, LPPOI
- 0	Modifies the window origin for a device context using the specified horizontal a
ScaleViewportExtEx	BOOL ScaleViewportExtEx(int xNum, int xDenom, int yNum int yDenom, LPSIZE lpSize) co
	Modifies the viewport for a device context using the ratios formed by the specifi

ScaleWindowExtEx	BOOL ScaleWindowExtEx(int xNum, int xDenom, int yNum, ir
	Modifies the window for a device context using the ratios formed by the specified n
SetMapMode	<pre>int SetMapMode(int nMapMode) const;</pre>
	Sets the mapping mode of the specified device context. The mapping mode defines units into device-space units, and also defines the orientation of the device's x and y
	BOOL SetViewportExtEx(int x, int y, LPSIZE lpSize) const
SetViewportExtEx	BOOL SetViewportExtEx(SIZE size, LPSIZE lpSizeRet) const
	Sets the horizontal and vertical extents of the viewport for a device context by using
	B00L SetViewportOrgEx(int x, int y, LPPOINT lpPoint) cor
SetViewportOrgEx	BOOL SetViewportOrgEx(POINT point, LPPOINT lpPointRet) c
	Specifies which window point maps to the viewport origin (0,0).
SetWindowExtEx	BOOL SetWindowExtEx(int x, int y, LPSIZE lpSize) const;
	BOOL SetWindowExtEx(SIZE size, LPSIZE lpSizeRet) const;
	Sets the horizontal and vertical extents of the window for a device context by using
SetWindowOrgEx	BOOL SetWindowOrgEx(int x, int y, LPPOINT lpPoint) const
	BOOL SetWindowOrgEx(POINT point, LPPOINT lpPointRet) cor
	Specifies which window point maps to the viewport origin (0,0).

#### **Printer Functions**

AbortDoc	<pre>int AbortDoc() const;</pre>
	Stops the current print job and erases everything drawn since the last call to the Star
EndDoc	<pre>int EndDoc() const;</pre>
	Ends a print job.
EndPage	<pre>int EndPage() const;</pre>
	Notifies the device that the application has finished writing to a page. This function advance to a new page.
SetAbortProc	<pre>int SetAbortProc( B00L (CALLBACK* lpfn)(HDC, int) ) cons</pre>
	Sets the application-defined abort function that allows a print job to be cancelled du

StartDoc	<pre>int StartDoc(LPD0CINF0 lpDocInfo) const;</pre>
	Starts a print job.
StartPage	<pre>int StartPage() const;</pre>
	Prepares the printer driver to accept data.
Font and Text Function	ns
DrawText	<pre>int DrawText(LPCTSTR lpszString, int nCount, const RECT&amp;</pre>
	Draws formatted text in the specified rectangle. It formats the text according to the scharacters, breaking lines, and so forth).
DrawTextEx	<pre>int DrawTextEx(LPTSTR lpszString, int nCount, const RECT LPDRAWTEXTPARAMS lpDTParams) const;</pre>
	Draws formatted text in the specified rectangle.
ExtTextOut	BOOL ExtTextOut(int x, int y, UINT nOptions, const RECT& UINT nCount, LPINT lpDxWidths) const;
	Draws text using the currently selected font, background color, and text color. You c clipping, opaquing, or both.
GetBkColor	COLORREF GetBkColor() const;
	Returns the current background color for the specified device context.
	<pre>int GetBkMode() const;</pre>
GetBkMode	Returns the current background mix mode for a specified device context. The backg hatched brushes, and pen styles that are not solid lines.
	BOOL GetCharABCWidths(UINT uFirstChar, UINT uLastChar, L
GetCharABCWidths	Retrieves the widths, in logical units, of consecutive characters in a specified range succeeds only with TrueType fonts.
	BOOL GetCharABCWidthsI(UINT giFirst, UINT cgi, LPWORD pg
GetCharABCWidthsI	Retrieves the widths, in logical units, of consecutive glyph indices in a specified ran succeeds only with TrueType fonts.
GetCharacterPlacement	DWORD GetCharacterPlacement(LPCTSTR pString, int nCount, LPGCP_RESULTS pResults, DWOR
	Retrieves information about a character string, such as character widths, caret positi rendering

GetCharWidth	BOOL GetCharWidth(UINT iFirstChar, UINT iLastChar, float
	Retrieves the fractional widths of consecutive characters in a specified range from tl
GetCharWidthI	BOOL GetCharWidthI(UINT giFirst, UINT cgi, LPWORD pgi, i
	Retrieves the widths, in logical coordinates, of consecutive glyph indices in a specif
GetFontData	DWORD GetFontData(DWORD dwTable, DWORD dwOffset, LPVOID
	Retrieves font metric data for a TrueType font.
GetFontLanguageInfo	<pre>DWORD GetFontLanguageInfo() const;</pre>
Gen onthanguagenno	Returns information about the currently selected font for the display context.
GetGlyphOutline	DWORD GetGlyphOutline(UINT uChar, UINT uFormat, LPGLYPHM LPVOID pvBuffer, CONST MAT2 *lpma
	Retrieves the outline or bitmap for a character in the TrueType font that is selected i
GetKerningPairs	DWORD GetKerningPairs(DWORD nNumPairs, LPKERNINGPAIR pkr
	Retrieves the widths, in logical coordinates, of consecutive glyph indices in a specif
	CSize GetTabbedTextExtent(LPCTSTR lpszString, int nCount LPINT lpnTabStopPositions) cor
GetTabbedTextExtent	Computes the width and height of a character string. If the string contains one or me upon the specified tab stops. The GetTabbedTextExtent function uses the currently s string.
GetTextAlign	UINT GetTextAlign() const;
GetTextruigh	Retrieves the text-alignment setting for the specified device context.
GetTextCharacterExtra	<pre>int GetTextCharacterExtra() const;</pre>
GetTextCharacterExtra	Retrieves the current intercharacter spacing for the specified device context.
GetTextColor	COLORREF GetTextColor() const;
GetTextColor	Retrieves the current text color for the specified device context.
GetTextExtentPoint32	CSize GetTextExtentPoint32(LPCTSTR lpszString, int nCour
	Computes the width and height of the specified string of text.
	<pre>int GetTextFace(int nCount, LPTSTR lpszFacename) const;</pre>
GetTextFace	Retrieves the typeface name of the font that is selected into the specified device con

GetTextMetrics	BOOL GetTextMetrics(TEXTMETRIC& Metrics) const;
	Fills the specified buffer with the metrics for the currently selected font.
GrayString	BOOL GrayString(HBRUSH hBrush, GRAYSTRINGPROC lpOutputFu int nCount, int x, int y, int nWidth, ir
	Draws gray text at the specified location. The function draws the text by copying it then copying the bitmap to the screen. The function grays the text regardless of the currently selected font.
	COLORREF SetBkColor(COLORREF crColor) const;
SetBkColor	Sets the current background color to the specified color value, or to the nearest phys color value.
	<pre>int SetBkMode(int iBkMode) const;</pre>
SetBkMode	Sets the background mix mode of the specified device context. The background mix styles that are not solid lines.
SetMapperFlags	<pre>DWORD SetMapperFlags(DWORD dwFlag) const;</pre>
11 0	Alters the algorithm the font mapper uses when it maps logical fonts to physical fon
SetTextAlign	UINT SetTextAlign(UINT nFlags) const;
	Sets the text-alignment flags for the specified device context.
	<pre>int SetTextCharacterExtra(int nCharExtra) const;</pre>
SetTextCharacterExtra	Sets the inter-character spacing. Inter-character spacing is added to each character, i a line of text.
SetTextColor	COLORREF SetTextColor(COLORREF crColor) const;
SetTextColor	Sets the text color for the specified device context to the specified color.
	<pre>int SetTextJustification(int nBreakExtra, int nBreakCour</pre>
SetTextJustification	Specifies the amount of space the system should add to the break characters in a stri calls the TextOut or ExtTextOut functions.
TabbedTextOut	CSize TabbedTextOut(int x, int y, LPCTSTR lpszString, ir LPINT lpnTabStopPositions, int nTabC
	Writes a character string at a specified location, expanding tabs to the values specifi in the currently selected font, background color, and text color.
TextOut	BOOL TextOut(int x, int y, LPCTSTR lpszString, int nCour
TextOut	Writes a character string at the specified location, using the currently selected font, l

## Remarks

CDC objects can be used anywhere a a handle to a device context (a HDC) might be used. They can be substituted for the HDC in any of the Windows API functions which use a HDC as a function argument. The benefit of using a CDC object is that it automatically deletes the device context when it is destroyed, along with any GDI objects created by the CDC.

There are a number of classes inherited from CDC, namely <u>CClientDC</u>, <u>CMemDC</u>, <u>CMetaFileDC</u>, <u>CPaintDC</u>, and <u>CWindowDC</u>. Typically one of these more specialized classes would be used to create the device context.

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

# **CDialog Class**

## Description

CDialog adds support for dialogs to Win32++. Dialogs are specialised windows which are a parent window for common controls. Common controls are special window types such as buttons, edit controls, tree views, list views, static text etc.

The layout of a dialog is typically defined in a resource script file (often Resource.rc). While this script file can be constructed manually, it is often created using a resource editor. If your compiler doesn't include a resource editor, you might find ResEdit useful. It is a free resource editor available for download at: <a href="http://www.resedit.net/">http://www.resedit.net/</a>

CDialog supports modal and modeless dialogs. It supports the creation of dialogs defined in a resource script file, as well as those defined in a dialog template.

The Dialog sample program can be used as the starting point for your own dialog applications.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of dialogs.

## **CDialog Members**

#### Construction

CDialog	CDialog(UINT nResID);
	CDialog(LPCTSTR lpszResName); CDialog(LPCDLGTEMPLATE lpTemplate);
	Constructor for CDialog. There are three forms of the CDialog constructor, one each for dialogs based on a resource ID, a resource string, and a dialog template.

#### **Overridables**

virtual BOOL DialogProc(UINT uMsg, WPARAM wParam, LPARAM `

DialogProc	The window procedure for this CDialog object. Override this function to specify how messages for this window are to be handled. Return all unhandled messages to DialogProcDefault.	
EndDialog	<pre>virtual void EndDialog(INT_PTR nResult); Ends a modal or modeless dialog.</pre>	
OnCancel	<pre>virtual void OnCancel(); Called when the Cancel button is pressed.</pre>	
OnInitDialog	<pre>virtual BOOL OnInitDialog(); Called when the dialog starts, before it is displayed. Override the function to specify y happens when the dialog is created.</pre>	
OnOK	<pre>virtual void OnOK(); Called when the OK button is pressed. The default behavior is to end the dialog when button is pressed.</pre>	
PreTranslateMessage	<pre>virtual BOOL PreTranslateMessage(MSG* pMsg); Override this function to filter mouse and keyboard messages prior to being passed to message loop.</pre>	

## Operations

AttachItem	<pre>virtual void AttachItem(int nID, CWnd&amp; Wnd);</pre>	
	Attach a dialog item to a CWnd.	
Create	<pre>virtual HWND Create(CWnd* pParent = NULL); Creates a modeless dialog.</pre>	
DialogProcDefault	virtual BOOL DialogProcDefault(UINT uMsg, WPARAM wParam, LP The default dialog procedure. All unhandled messages should be passed to this function.	
DoModal	<pre>virtual INT_PTR DoModal(CWnd* pParent = NULL); Create a modal dialog. A modal dialog must be closed before the owner window can be i</pre>	
DoModeless	<pre>virtual HWND DoModeless(CWnd* pParent = NULL); Create a modeless dialog.</pre>	

GetDefID	DWORD GetDefID() const;	
	Retrieves the identifier of the default push button control for the dialog.	
GotoDlgCtrl	<pre>void GotoDlgCtrl(CWnd* pWndCtrl);</pre>	
	Sets the keyboard focus to the specified control.	
MapDialogRect	BOOL MapDialogRect(LPRECT pRect) const;	
	Converts the dialog box units to screen units (pixels).	
NextDlgCtrl	<pre>void NextDlgCtrl() const;</pre>	
	Sets the keyboard focus to the next dialog control.	
PrevDlgCtrl	<pre>void PrevDlgCtrl() const;</pre>	
	Sets the keyboard focus to the previous dialog control.	
SetDefID	<pre>void SetDefID(UINT nID);</pre>	
	Changes the identifier of the default push button for a dialog box.	

## **State functions**

IsModal	BOOL IsModal() const; Returns TRUE of the dialog is modal.
IsIndirect	BOOL IsIndirect() const; Returns TRUE of the dialog is created from a dialog box template in memory.

## **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Modal dialogs are always the application's active window. While it is are running, the modal dialog is the only window that can accept user input for the application. Modeless dialogs, on the other hand, allow other windows to become active and accept user input while they are running.

Modal dialogs run their own internal message loop, whereas modeless dialogs run the message loop provided by CWinApp. While the modal dialog is running, its internal message loop processes all messages for the thread the dialog runs in, including those of other windows.

Dialogs are used to display various controls, such as buttons, TreeViews, ListViews, static text and so forth. Each of these controls is actually a child window of the dialog. The following function are often used to manage dialog controls:

- AttachDlgItem
- CheckDlgButton
- GetDlgItem
- GetDlgItemInt
- GetDlgItemText
- SendDlgMessage
- SetDlgItemInt
- SetDlgItemText

The AttachDlgItem function is used to attach a dialog control to <u>CWnd</u> derived object. This allows the CWnd derived object to handle the control's messages in WndProc, and the control's notifications in OnNotifyReflect.

A modeless dialog can be used as a view window for a frame, MDIChild, docker etc. The FormDemo sample demonstrate how to use a dialog as a view window in a <u>CFrame</u>. This sample also demonstrates how to use <u>CResizer</u> to automatically reposition the dialog's controls when the frame is resized.

Refer to the <u>Dialog</u> section for more information on how to use CDialog to create dialogs.

Header file	dialog.h
Win32/64 support	Yes
WinCE support	Yes

# **CDocker Class**

## Description

The CDocker class adds both docking and splitter windows to the Win32++ framework. Splitter windows have a subset of the functionality of docking windows. Splitter windows are child windows that can be dynamically re-sized. Docking windows have this feature too, but they also allow windows to be docked and undocked.

## **CDocker Members**

CDocker ();

Constructor for CDocker.

## AddDockedChild

Adds the specified Docker as a docked child. AddUndockedChild

Adds the specified Docker as an undocked member of this docking group. Close

virtual void Close();

Closes the Docker. CloseAllDockers

virtual void CloseAllDockers();

Closes all Dock children. Dock

virtual void Dock(CDocker\* pDocker, UINT uDockSide);

Docks the Docker to the side of another Docker. DockInContainer

virtual void DockInContainer(CDocker\* pDock, DWORD dwDockStyle);

Docks the Container inside another Container (by adding a tab). GetActiveDocker

virtual CDocker\* GetActiveDocker() const;

Returns the docker whose child window has focus. GetAllDockers

std::vector <CDocker\*> \* GetAllDockers() const;

Returns a pointer to the vector containing pointers to all Dockers in this docking group. GetBarWidth

int GetBarWidth() const;

Returns the width of the Docker's splitter bar. GetCaption

CString& GetCaption() const;

Returns the Docker's caption text. GetContainer

virtual CDockContainer\* GetContainer() const;

Returns a pointer to the container used as the Docker's view. If the view is not a container, this function returns NULL. GetDockAncestor

virtual CDocker\* GetDockAncestor() const;

Returns a pointer to the Dock Ancestor of this docking group. GetDockBar

virtual CDockBar\* GetDockBar() const;

Returns a pointer to this docker's splitter bar. GetDockChildren

std::vector <CDocker\*> \* GetDockChildren() const;

Returns a pointer to the vector containing pointers to all of this docker's dock children. GetDockClient

virtual CDockClient\* GetDockClient() const;

Returns a pointer to the Window of this docker's client area. The dock caption an view window are child windows of the dock client. GetDockFromID

virtual CDocker\* GetDockFromID(int n\_DockID) const;

Returns a pointer to the docker, given its ID. GetDockFromPoint

#### virtual CDocker\* GetDockFromPoint(POINT pt) const;

Returns a pointer to the Docker whose Dock client area includes the specified point . GetDockFromView

virtual CDocker\* GetDockFromView(CWnd\* pView) const;

Returns a pointer to the Docker which has the the specified view window. GetDockHint

virtual CDockHint\* GetDockHint() const;

Returns a pointer to the Docker's hint window. The hint window as a blue tint and is displayed during dock dragging. GetDockID

int GetDockID() const;

Return the ID of this Docker. GetDockParent

CDocker\* GetDockParent() const;

Returns the Dock Parent of this Docker. GetDockStyle

DWORD GetDockStyle() const;

Returns the dock style of this Docker. GetDockWidth

virtual int GetDockWidth() const;

Returns the width of a docker which is dock to the left or right of another docker, or its height if docked to the top or bottom. GetTabbedMDI

virtual CTabbedMDI\* GetTabbedMDI() const;

Returns a TabbedMDI pointer to the view window of this docker, or NULL if the view window is not a TabbedMDI. **GetTopmostDocker** 

virtual CDocker\* GetTopmostDocker() const;

Returns the top level Dock parent of this Docker. Any undocked Docker (not just the Dock Ancestor) can be a top level Dock parent. GetView

CWnd\* GetView() const;

Returns a pointer to the view window of this docker. GetViewRect

virtual CRect GetViewRect() const;

Returns a CRect containing the dimensions of the view window.  $\operatorname{Hide}$ 

virtual void Hide();

Undocks the Docker (if necessary) and hides it. IsChildOfDocker

BOOL IsChildOfDocker(HWND hwnd) const;

Returns TRUE if the specified window handle is a decendant of this docker. Is Docked

BOOL IsDocked() const;

Returns TRUE if this Docker is docked within another Docker. IsDragAutoResize

B00L IsDragAutoResize() const;

Returns the TRUE of the dockers will be automatically re-arrange as the splitter bar is moved. **IsRelated** 

BOOL IsRelated(HWND hWnd) const;

Returns TRUE if the specified Docker is in this docker family. Dockers in the same dock family share the one dock ancestor. IsUndocked

BOOL IsUndocked() const;

Returns TRUE if the docker is Undocked. The dock ancestor is always undocked, but other dockers can dock within it. LoadDockRegistrySettings

virtual BOOL LoadDockRegistrySettings(LPCTSTR szRegistryKeyName);

Adds dockers to the dock ancestor according to the docking information stored in the registry. Recalc Dock Layout

virtual void RecalcDockLayout();

Recalculates and repositions all dockers which are dock decendants of this docker's top level dock parent. SaveDockRegistrySettings

virtual BOOL SaveDockRegistrySettings(LPCTSTR szRegistryKeyName);

Saves the docking styles, state and position in the registry. SetBarColor

void SetBarColor(COLORREF color);

Sets the color of the splitter bar. SetBarWidth

#### void SetBarWidth(int nWidth);

Sets the width of the splitter. SetCaption

void SetCaption(LPCTSTR szCaption);

Sets the dock caption text. SetCaptionColors

vvoid SetCaptionColors(COLORREF Foregnd1, COLORREF Backgnd1, COLORREF Fo

Sets the caption's foreground and background colours. SetCaptionHeight

void SetCaptionHeight(int nHeight);

Sets the caption's height. SetDockStyle

void SetDockStyle(DWORD dwDockStyle);

Set the dock style. SetDockWidth

void SetDockWidth(int DockWidth);

Sets the width of a docker docked to the left or right of its parent, or the height of the docker docked to the top or bottom. SetDragAutoResize

void SetDragAutoResize(BOOL bAutoResize);

Set the DragAutoSize mode. When TRUE, dockers will be automatically re-arranged as the splitter bar is moved. When FALSE the dockers will be re-arranged when the splitter bar dragging is complete. SetView

void SetView(CWnd& wndView);

Sets the view window for the docker. Undock

virtual void Undock(CPoint pt, BOOL bShowUndocked = TRUE);

Undocks the Docker and positions it at the specified point. UndockContainer

Undocks the Container and positions it at the specified point. VerifyDockers

virtual BOOL VerifyDockers();

A built in diagnostic which verifies the integrity of the docking hierarchy.

## Overridables

NewDockerFromID	<pre>virtual CDocker* NewDockerFromID(int idDock); Override this function to create a new Docker given its docker ID.</pre>
OnActivate	<pre>virtual void OnActivate(WPARAM wParam, LPARAM lParam); Called when the window is activated or deactivated.</pre>
OnBarEnd	<pre>virtual LRESULT OnBarEnd(LPDRAGPOS pdp); Called when the repositioning of the splitter bar is complete.</pre>
OnBarMove	<pre>virtual LRESULT OnBarMove(LPDRAGPOS pdp); Called when the splitter bar is moved.</pre>
OnBarStart	<pre>virtual LRESULT OnBarStart(LPDRAGPOS pdp); Called when the splitter bar is about to be repositioned.</pre>
OnCreate	<pre>virtual void OnCreate(); Called when the window is created.</pre>
OnDestroy	<pre>virtual void OnDestroy(WPARAM wParam, LPARAM lParam); Called when the window is destroyed.</pre>
OnDockActivated	<pre>virtual LRESULT OnDockActivated(WPARAM wParam, LPARAM l Called when a docker is activated or deactivated.</pre>
OnDockDestroyed	virtual void OnDockDestroyed(WPARAM wParam, LPARAM lPar Called when the docker is destroyed.
OnDockEnd	<pre>virtual LRESULT OnDockEnd(LPDRAGPOS pdp); Called when docker has been docked.</pre>
OnDockMove	<pre>virtual LRESULT OnDockMove(LPDRAGPOS pdp); Called when an undocked docker is being moved.</pre>
OnDockSetFocus	<pre>virtual LRESULT OnDockSetFocus(); Called when a child docker gets focus.</pre>
	<pre>virtual LRESULT OnDockStart(LPDRAGPOS pdp);</pre>

OnDockStart	Called when undocking is about to start.	
OnExitSizeMove	virtual void OnExitSizeMove(WPARAM wParam, LPARAM lPara Called after the window has completed a resize or move.	
OnNotify	<pre>virtual LRESULT OnNotify(WPARAM wParam, LPARAM lParam); Called when the window receives a notification.</pre>	
OnNCLButtonDblClk	virtual LRESULT OnNCLButtonDblClk(WPARAM wParam, LPARAM Called when the non-client area recieves a left button double click.	
OnSysColorChange	virtual void OnSysColorChange(WPARAM wParam, LPARAM lPa Called when a change is made to a system color setting.	
OnSysCommand	virtual LRESULT OnSysCommand(WPARAM wParam, LPARAM lPar Called when the window is minimized, maximized, restored or closed.	
OnTimer	<pre>virtual LRESULT OnTimer(WPARAM wParam, LPARAM lParam); Called when the timer has expired.</pre>	
OnWindowPosChanged	virtual void OnWindowPosChanged(WPARAM wParam, LPARAM l Called after the window position has changed.	
OnWindowPosChanging	virtual LRESULT OnWindowPosChanging(WPARAM wParam, LPAR Called before the window position has changed.	
PreCreate	<pre>virtual void PreCreate(CREATESTRUCT &amp;cs); Sets the window creation parameters.</pre>	
PreRegisterClass	<pre>virtual void PreRegisterClass(WNDCLASS &amp;wc); Sets the window class parameters.</pre>	
PreTranslateMessage	<pre>virtual BOOL PreTranslateMessage(MSG Msg); This functions is called by the MessageLoop. It processes the keyboard accelerator CWnd::PreTranslateMessage for keyboard and mouse events.</pre>	

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

# Remarks

Refer to the <u>Docking</u> section for information on how to use CDocker to create splitter windows and docking windows.

Header file	docking.h
Win32/64 support	Yes
WinCE support	Yes

# **CDockContainer Class**

# Description

The CDockContainer class adds "Containers" to the Win32++ framework. A Container is a tab control which has been designed to co-operate with docking. While Dockers allows any child window to be used as the view window, Containers add additional features when used as the view window for Dockers. These additional features include Container within Container docking (where the newly docked Container adds another tab), as well as additions to the dock targeting and dock hinting visual cues. Containers can also have a toolbar. The use of a toolbar is optional, but when used they are set up in the same way as toolbars for Frames.

CDockContainers have views of their own. These views can be any child window, and are set in the same way as views for Frames and MDI children.

## **CDockContainer Members**

CDockContainer();

Constructor for CDockContainer.

# AddContainer

virtual void AddContainer(CDockContainer\* pContainer, BOOL bInsert = FAL

Adds the specified container as a child to this container. Set bInsert to TRUE to insert the container as the first tab, or FALSE to add it as the last tab. AddToolBarButton

virtual void AddToolBarButton(UINT nID, BOOL bEnabled = TRUE);

Adds a resource ID to the container's toolbar. GetActiveContainer

CDockContainer\* GetActiveContainer() const;

Returns a pointer to the currently active container. GetActiveView

CWnd\* GetActiveView() const;

Returns a pointer to the view window for the currently active container. GetAllContainers

std::vector<ContainerInfo>\* GetAllContainers() const;

Returns a reference to the vector of container information. GetContainerFromIndex

#### virtual CDockContainer\* GetContainerFromIndex(UINT nPage);

Returns a pointer to the container at the specified tab number. GetContainerFromView

virtual CDockContainer\* GetContainerFromView(CWnd\* pView) const;

Returns a pointer to the container with the specified view. GetContainerIndex

virtual int GetContainerIndex(CDockContainer\* pContainer);

Returns the tab index of the specified container. GetContainerParent

CDockContainer\* GetContainerParent() const;

Returns a pointer to the container which is the parent of this container group. GetDockCaption

CString& GetDockCaption() const;

Returns the CString which contains the text displayed in the caption of a docked container. GetMaxTabTextSize

virtual SIZE GetMaxTabTextSize();

Returns the size (width and height) of the caption text. GetTabIcon

HICON GetTabIcon() const;

Returns the icon handle for this container's tab. GetTabText

LPCTSTR GetTabText() const;

Returns the text for this container's tab. GetToolBar

virtual CToolBar\* GetToolBar() const;

Returns a pointer to the container's toolbar. GetView

CWnd\* GetView() const;

Returns a pointer to the view window for this container. GetViewPage

virtual CViewPage\* GetViewPage() const;

Returns a pointer to the container's page. The container's page holds the toolbar and view window.

# RecalcLayout

#### virtual void RecalcLayout();

Recalculates the positions of the child windows for the container, and repositions them. **RemoveContainer** 

virtual void RemoveContainer(CDockContainer\* pWnd);

Removes the specified child container from this container group. SelectPage

virtual void SelectPage(int nPage);

Activates the specified page number. SetActiveContainer

void SetActiveContainer(CDockContainer\* pContainer);

Sets the active container. SetDockCaption

void SetDockCaption(LPCTSTR szCaption);

Sets the text associated with a docked container. SetHideSingleTab

void SetHideSingleTab(BOOL bHide);

Shows or hides the tab if it has only one page.  $\ensuremath{\textit{SetTabIcon}}$ 

void SetTabIcon(HICON hTabIcon);

void SetTabIcon(UINT nID\_Icon);

Sets the icon for this container's tab.  $\ensuremath{\textit{SetTabText}}$ 

void SetTabText(LPCTSTR szText);

void SetTabText(UINT nTab, LPCTSTR szText);

Sets the text for this container's tab. SetToolBarImages

virtual void SetToolBarImages(COLORREF crMask, UINT ToolBarID, UINT Tool

Sets the normal, hot and disabled images for the ToolBar SetupToolBar

virtual void SetupToolBar();

Override this function to specify the container's toolbar's bitmap and resource IDs. SetView

#### void SetView(CWnd& Wnd);

## Sets the view window for this container. SwapTabs

#### virtual void SwapTabs(UINT nTab1, UINT nTab2);

Swaps the position of two tabs.

#### Overridables

OnLButtonDown	OnLButtonDown(WPARAM wParam, LPARAM lParam); Called when the left mouse button is pressed.		
OnLButtonUp	OnLButtonUp(WPARAM wParam, LPARAM lParam); Called when the left mouse button is released.		
OnMouseLeave	<pre>virtual LRESULT OnMouseLeave(WPARAM wParam, LPARAM lParam); Called when the mouse cursor leaves the window.</pre>		
OnMouseMove	<pre>virtual LRESULT OnMouseMove(WPARAM wParam, LPARAM lParam); Called when the mouse cursor is moved over the window.</pre>		
OnSetFocus	<pre>virtual LRESULT OnSetFocus(WPARAM wParam, LPARAM lParam); Called when the window gets keyboard focus. We set the keyboard focus to the active view window.</pre>		
OnSize	<pre>virtual LRESULT OnSize(WPARAM wParam, LPARAM lParam); Called when the window is resized.</pre>		
OnTCNSelChange	<pre>virtual LRESULT OnTCNSelChange(LPNMHDR pNMHDR); Called when the currently selected tab has changed.</pre>		

#### **Base class Members**

For base class members, refer to the members of <u>CTab</u>.

### Remarks

The CDockContainer could be also used as a view window for other window types, such as Frames and MDI children, although using a CTab for these other

views is more common.

Header file	docking.h
Win32/64 support	Yes
WinCE support	No
Library required	Comctl32.lib

# **CEdit Class**

## Description

An edit control is a rectangular control window typically used in a dialog box to permit the user to enter and edit text by typing on the keyboard.Edit controls are typically used in dialog boxes, but you can use them in the client area of a standard window as well. Single-line edit controls are useful for retrieving a single string from the user. Multiline edit controls make it easy for your application to implement most of the features of a simple word processor.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of edit controls.

# **CEdit Members**

#### Construction

CEdit	CEdit();
	Constructor for CEdit.

### Attributes

	<pre>int GetFirstVisibleLine() const;</pre>
GetFirstVisibleLine	
	based index of the uppermost visible line in a multiline edit control.
	HLOCAL GetHandle() const;
GetHandle	
	Returns a handle identifying the buffer containing the multiline edit control's text. It is n processed by single-line edit controls.
GetLimitText	UINT GetLimitText() const;
	Returns the current text limit, in characters.
	<pre>int GetLine(int nIndex, LPTSTR lpszBuffer) const;</pre>
GetLine	int deterne(int hindex, Erisik (pszbarrer) const,
	<pre>int GetLine(int nIndex, LPTSTR lpszBuffer, int nMaxLength)</pre>
Gettine	

	Copies characters in a single-line edit control to a buffer and returns the number of copied. In a multiline edit control, retrieves a line of text from the control and return of characters copied.
GetLineCount	<pre>int GetLineCount() const;</pre>
	Returns the number of lines in the edit control.
GetMargins	DWORD GetMargins() const;
0	Returns the widths of the left and right margins.
GetModify	BOOL GetModify() const;
	Returns a flag indicating whether the content of an edit control has been modified.
GetPasswordChar	TCHAR GetPasswordChar() const;
	Returns the character that edit controls use in conjunction with the ES_PASSWOR
GetRect	<pre>void GetRect(LPRECT lpRect) const;</pre>
	Returns the coordinates of the formatting rectangle in an edit control.
GetSel	<pre>void GetSel(int&amp; nStartChar, int&amp; nEndChar) const;</pre>
	Returns the starting and ending character positions of the current selection in the e
	<pre>void SetHandle(HLOCAL hBuffer) const;</pre>
SetHandle	Sets a handle to the memory used as a text buffer, empties the undo buffer, resets the positions to zero, and redraws the window.
SetLimitText	<pre>void SetLimitText(UINT nMax) const;</pre>
	Sets the maximum number of characters the user may enter in the edit control.
	<pre>void SetMargins(UINT nLeft, UINT nRight) const;</pre>
SetMargins	Sets the widths of the left and right margins, and redraws the edit control to reflect margins.
SetModify	<pre>void SetModify(BOOL bModified = TRUE) const;</pre>
	Sets or clears the modification flag to indicate whether the edit control has been mo
SetPasswordChar	<pre>void SetPasswordChar(TCHAR ch) const;</pre>
	Defines the character that edit controls use in conjunction with the ES_PASSWOR
SetReadOnly	B00L SetReadOnly(B00L bReadOnly = TRUE) const;

	Sets or removes the read-only style (ES_READONLY) in an edit control.
SetRect	<pre>void SetRect(LPCRECT lpRect) const;</pre>
	Sets the formatting rectangle for the multiline edit control and redraws the window. It is processed by single-line edit controls.
SetRectNP	<pre>void SetRectNP(LPCRECT lpRect) const;</pre>
	Sets the formatting rectangle for the multiline edit control but does not redraw the wind not processed by single-line edit controls.
	<pre>void SetSel(DWORD dwSelection, BOOL bNoScroll) const;</pre>
SetSel	<pre>void SetSel(int nStartChar, int nEndChar, BOOL bNoScroll) (</pre>
	Selects a range of characters in the edit control by setting the starting and ending positio selected.
SetTabStops	BOOL SetTabStops(int nTabStops, LPINT rgTabStops) const;
	BOOL SetTabStops(const int& cxEachStop) const;
	BOOL SetTabStops() const;
	Sets tab-stop positions in the multiline edit control. It is not processed by single-line edi

# Operations

CanUndo	BOOL CanUndo() const; Returns TRUE if the edit control operation can be undone.	
CharFromPos	<pre>int CharFromPos(CPoint pt) const; Returns the character index and line index of the character nearest the specified point.</pre>	
Clear	<pre>void Clear() const; Clears the current selection, if any, in an edit control. If there is no current selection, deletes the character to the right of the caret.</pre>	
Сору	<pre>void Copy() const; Copies text to the clipboard unless the style is ES_PASSWORD, in which case the message returns zero.</pre>	
Cut	<pre>void Cut() const; Cuts the selection to the clipboard, or deletes the character to the left of the cursor if</pre>	

	there is no selection.
EmptyUndoBuffer	<pre>void EmptyUndoBuffer() const;</pre>
	Empties the undo buffer and sets the undo flag retrieved by the EM_CANUNDO message to FALSE. The system automatically clears the undo flag whenever the econtrol receives a WM_SETTEXT or EM_SETHANDLE message.
	B00L FmtLines(B00L bAddE0L) const;
FmtLines	Adds or removes soft line-break characters (two carriage returns and a line feed) to ends of wrapped lines in a multiline edit control. It is not processed by single-line controls.
	<pre>void LimitText(int nChars = 0) const;</pre>
LimitText	Sets the text limit of an edit control. The text limit is the maximum amount of text, TCHARs, that the user can type into the edit control.
	<pre>int LineFromChar(int nIndex = -1) const;</pre>
LineFromChar	Returns the zero-based number of the line in a multiline edit control that contains a specified character index. This message is the reverse of the EM_LINEINDEX message. It is not processed by single-line edit controls.
	<pre>int LineIndex(int nLine = -1) const;</pre>
LineIndex	Returns the character of a line in a multiline edit control. This message is the rever the EM_LINEFROMCHAR message. It is not processed by single-line edit contro
	<pre>int LineLength(int nLine = -1) const;</pre>
LineLength	Returns the length, in characters, of a single-line edit control. In a multiline edit correturns the length, in characters, of a specified line.
	<pre>void LineScroll(int nLines, int nChars = 0) const;</pre>
LineScroll	Scrolls the text vertically in a single-line edit control or horizontally in a multiline control (when the control has the ES_LEFT style). The lParam parameter specifies number of lines to scroll vertically, starting from the current line. The wParam parameter specifies the number of characters to scroll horizontally, starting from the current character.
Dasta	<pre>void Paste() const;</pre>
Paste	Pastes text from the clipboard into the edit control window at the caret position.
PosFromChar	CPoint PosFromChar(UINT nChar) const;
	Returns the client coordinates of the specified character.

ReplaceSel	Replaces the current selection with the text in an application-supplied buffer, sends the parent window EN_UPDATE and EN_CHANGE messages, and updates the undo buffer.
Undo	<pre>void Undo() const; Removes any text that was just inserted or inserts any deleted characters and sets the selection to the inserted text. If necessary, sends the EN_UPDATE and EN_CHANGE</pre>
	notification messages to the parent window.

### Overidables

PreRegisterClass	<pre>virtual void PreRegisterClass(WNDCLASS &amp;wc);</pre>	
	Set the window class parameters before the window is created.	

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Like all common controls, the edit control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for an edit control.

Header file	stdcontrols.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CFile Class**

## Description

This class manages the reading from and writing to files.

## **CFile Members**

Initialisation and Assignment	
	CFile();
CFile	CFile(HANDLE hFile);
	<pre>CFile(LPCTSTR pszFileName, UINT n0penFlags);</pre>
	Constructor for CFile.

Attributes GetFileName

virtual const CString& GetFileName() const;

Returns the filename of the file associated with this object. GetFilePath

virtual const CString& GetFilePath() const;

Returns the full filename including the directory of the file associated with this object. GetFileTitle

virtual const CString& GetFileTitle() const;

Returns the filename of the file associated with this object, excluding the path and the file extension. GetHandle

HANDLE GetHandle() const;

Converts the CFile to a HANDLE. GetPosition

virtual ULONGLONG GetPosition() const;

Returns the current value of the file pointer, which can be used in subsequent calls to Seek. Operator HANDLE

operator HANDLE() const;

Converts the CFile to a HANDLE. **Operations** Close

virtual BOOL Close();

Closes the file associated with this object. Closed file can no longer be read or written to.. Flush

virtual BOOL Flush();

Causes any remaining data in the file buffer to be written to the file. LockRange

virtual BOOL LockRange(ULONGLONG Pos, ULONGLONG Count);

Locks a range of bytes in and open file. Open

virtual int CollateNoCase(LPCTSTR pszText) const;

Prepares a file to be written to or read from. OpenFileDialog

#### virtual CString OpenFileDialog(LPCTSTR pszFilePathName = NULL, DWORD dwFlags = OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT, LPCTSTR pszTi LPCTSTR pszFilter = NULL, CWnd\* pOwnerWnd = NULL);

Displays the file open dialog. Returns a CString containing either the selected file name or an empty CString. Read

virtual UINT Read(void\* pBuf, UINT nCount);

Reads from the file, storing the contents in the specified buffer. Remove

static BOOL Remove(LPCTSTR pszFileName);

Deletes the specified file. Rename

static BOOL Rename(LPCTSTR pszOldName, LPCTSTR pszNewName);

Renames the specified file. SaveFileDialog

virtual CString SaveFileDialog(LPCTSTR pszFilePathName = NULL, DWORD dwFlags = OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT, LPCTSTR pszTi LPCSTR pszFilter = NULL, LPCTSTR pszDefExt = NULL, CWnd\* pOwnerWnd =

Displays the SaveFileDialog. Returns a CString containing either the selected file name or an empty CString. Seek

virtual ULONGLONG Seek(LONGLONG lOff, UINT nFrom);

Positions the current file pointer. Permitted values for nFrom are: FILE\_BEGIN, FILE\_CURRENT, or FILE\_END. SeekToBegin

virtual void SeekToBegin();

Sets the current file pointer to the beginning of the file. SeekToEnd

virtual ULONGLONG SeekToEnd();

Sets the current file pointer to the end of the file. SetFilePath

```
virtual void SetFilePath(LPCTSTR pszNewName);
```

Specifies the full file name, including its path SetLength

virtual BOOL SetLength(ULONGLONG NewLen);

Changes the length of the file to the specified value. UnlockRange

```
virtual BOOL UnlockRange(ULONGLONG Pos, ULONGLONG Count);
```

Unlocks a range of bytes in an open file. Write

virtual BOOL Write(const void\* pBuf, UINT nCount);

Writes the specified buffer to the file.

### Remarks

The following code demonstrates how to use OpenDileDialog to retrieve the name of the file to open.

```
void CMainFrame::OnFileOpen()
{
    CFile File;
    CString str = File.OpenFileDialog(0, OFN_FILEMUSTEXIST, _T("Scribble F
    if (!str.IsEmpty())
    {
        // Retrieve the PlotPoint data
        m_View.FileOpen(str);
    }
}
```

The following code demonstrates how to open a file for reading, and read its contents.

```
BOOL CView::FileOpen(LPCTSTR szFilename)
{
    DWORD nBytesRead;
    BOOL bResult = FALSE;
```

```
// Create a handle to the file
CFile File;
if (File.Open(szFilename, OPEN_EXISTING))
{
    do
    {
        nBytesRead = 0;
        PlotPoint pp;
        nBytesRead = File.Read(&pp;, sizeof(PlotPoint));
        if (nBytesRead == sizeof(PlotPoint))
            m_points.push_back(pp);
    } while (nBytesRead == sizeof(PlotPoint));
}
```

Header file	file.h
Win32/64 support	Yes
WinCE support	Yes

# **CFont Class**

## Description

The class responsible for creating and managing font resources.

# **CFont Members**

Initialization and Assignment			
	CFont();		
CFont	CFont(HFONT hFont);		
	<pre>CFont(const LOGFONT* lpLogFont);</pre>		
	Constructs a CFont object.		

## FromHandle

static CFont\* FromHandle(HFONT hFont);

Returns the CFont associated with the specified font handle. If a CFont object doesn't already exist, a temporary CFont object is created. This temporary CFont will be deleted sometime after the processing of

the current message is complete. **Operator HFONT()** 

operator HFONT() const;

Allows a CFont object to be used as a font handle (HFONT). Attributes GetLogFont

LOGFONT GetLogFont() const;

Retrieves the LOGFONT structure that contains font attributes. Operations CreateFont

HFONT CreateFont(int nHeight, int nWidth, int nEscapement, int nOrientation, int nWeight, DWORD dwItalic, DWORD dwUnderline, DWORD dwStrikeOut, DWORD dwCharSet, DWORD dwOutPrecision, DWORD dwClipPrecision, DWORD dwQua DWORD dwPitchAndFamily, LPCTSTR lpszFacename); Creates a logical font with the specified characteristics. CreateFontIndirect

HFONT CreateFontIndirect(const LOGFONT\* lpLogFont);

Creates a logical font that has the characteristics specified in the LOGFONT struct. CreatePointFont

HFONT CreatePointFont(int nPointSize, LPCTSTR lpszFaceName, CDC\* pDC = N BOOL bBold = FALSE, BOOL bItalic = FALSE);

Creates a font of a specified typeface and point size. CreatePointFontIndirect

HFONT CreatePointFontIndirect(const LOGFONT\* lpLogFont, CDC\* pDC = NULL)

Creates a font of a specified typeface and point size. This function automatically converts the height in the LOGFONT's lfHeight to logical units using the specified device context.

#### **Base class Members**

For base class members, refer to the members of <u>CGDIObject</u>.

### Remarks

CFont objects can be used anywhere a a handle to a font (HFONT) might be used. They can be substituted for the HFONT in any of the Windows API functions which use a HFONT as a function argument. The benefit of using a CFont object is that it automatically deletes the font when it is destroyed.

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

# **CFrame Class**

## Description

CFrame is responsible for creating a window which includes a menu and toolbar (usually hosted within a rebar), and a status bar, and the "view" window. The "view" window is a separate <u>CWnd</u> object assigned to the frame with the SetView function. This view window can be any sort of window, as long as it can be a child window of the frame. CFrame positions the view window over the part of the frame's client client area that remains after the toolbar and statusbar have been displayed.

To create a frame application, inherit a CMainFrame class from CFrame. Use the Frame sample as the starting point for your own frame applications. Refer to the Notepad and Scribble samples for examples on how to use these classes to create a frame application.

CFrame provides support for a single "view" window. This type of frame application is called a Single Document Interface or SDI. A Multiple Document Interface (or MDI) application allows several view windows to be displayed concurrently. Refer to the <u>CMDIFrame</u> class if your application requires several view windows to be displayed.

## **CFrame Members**

#### Construction

CFrame	CFrame();
	Constructs a CFrame object.

#### Attributes

GetFrameMenu       HMENU GetFrameMenu() const;         Returns the menu handle for the frame.         virtual int GetMenuItemPos(HMENU hMenu, LPCTSTR szItem);	GetFrameAccel	HACCEL GetFrameAccel() const; Returns the accelerator handle for the frame.
virtual int GetMenuItemPos(HMENU hMenu, LPCTSTR szItem):	GetFrameMenu	
GetMenultemPos       Returns the position of the menu item, given it's name.	GetMenuItemPos	<pre>virtual int GetMenuItemPos(HMENU hMenu, LPCTSTR szItem); Returns the position of the menu item, given it's name.</pre>

GetMenuBar	virtual CMenuBar* GetMenuBar() const;
GetivienuDu	Returns a pointer to the <u>MenuBar</u> . The MenuBar's window is a child of the reb which in turn is a child of the frame. The MenuBar displays a menu inside the r
GetMenuBarTheme	<pre>MenuTheme* GetMenuBarTheme() const;</pre>
	Returns a pointer to the MenuTheme structure used by the frame.
GetMRUEntry	CString GetMRUEntry(UINT nIndex);
5	Returns a the MRU string at the specified index.
GetReBar	virtual CReBar* GetReBar() const;
	Returns a pointer to the <u>ReBar</u> . The ReBar's window is a child of the frame.
GetReBarTheme	ReBarTheme* GetReBarTheme() const;
	Returns a pointer to the ReBarTheme structure used by the frame.
GetRegistryKeyName	CString GetRegistryKeyName() const;
	Returns the name of the registry key used by the applications to save its setting
GetStatusBar	virtual CStatusBar* GetStatusBar() const;
	Returns a pointer to the <u>StatusBar</u> . The StatusBar's window is a child of the frame
GetStatusBarTheme	<pre>StatusBarTheme* GetStatusBarTheme() const;</pre>
	Returns a pointer to the StatusBarTheme structure used by the frame.
GetStatusText	CString GetStatusText() const
Getotutus Tent	Returns the text displayed in the status bar.
GetThemeName	<pre>virtual CString GetThemeName() const;</pre>
Germemer tunic	Returns the name of the current XP theme name.
GetTitle	CString GetTitle() const;
	Returns the caption (window tiltle) of the frame window.
	virtual CToolBar* GetToolBar() const;
GetToolBar	Returns a pointer to the <u>ToolBar</u> . The ToolBar's window is a child of either the rebar or the frame.
GetToolBarTheme	ToolBarTheme* GetToolBarTheme() const;

	Returns a pointer to the ToolBarTheme structure used by the frame.
GetView	CWnd* GetView() const;
	Returns a pointer to the <u>CWnd</u> object positioned over the client area of the frame, commonly referred to as the View window.
	<pre>virtual CRect GetViewRect() const;</pre>
GetViewRect	Returns the coordinates of the View window. The View window is positioned over the part of the client area of the frame that remains after the toolbar and statusbar have been displayed.
	<pre>void SetFrameMenu(INT ID_MENU);</pre>
SetFrameMenu	<pre>void SetFrameMenu(HMENU hMenu);</pre>
	Sets the menu handle for the frame.
SetMenuTheme	<pre>void SetMenuTheme(MenuTheme* pMBT);</pre>
	Sets the theme colors used when menu items are selected.
SetReBarTheme	<pre>void SetReBarTheme(ReBarTheme* pRBT);</pre>
	Sets the theme colors used for the ReBar.
SetStatusBarTheme	<pre>void SetStatusBarTheme(StatusBarTheme* pSBT);</pre>
	Sets the theme colors used for the StatusBar.
SetStatusText	<pre>void SetStatusText(LPCTSTR szText);</pre>
	Sets the text to be displayed in the status bar.
SetTitle	<pre>void SetTitle(LPCTSTR szText)</pre>
	Sets the title (caption) of the frame window.
SetToolBarTheme	<pre>void SetToolBarTheme(ToolBarTheme* pTBT);</pre>
	Sets the theme colors used for the ToolBar.
SatViou	<pre>void SetView(CWnd&amp; wndView);</pre>
SetView	Sets the <u>CWnd</u> object which will be positioned over the client area of the frame. This is CWnd object is referred to as the view window.

# State functions

IsMenuBarUsed	BOOL IsMenuBarUsed() const;
	Returns TRUE if the frame is using a menubar.
IsMDIFrame	virtual BOOL IsMDIFrame() const;
	Returns TRUE if the frame is a MDI Frame. Refer to <u>CMDIFrame</u> .
IsReBarSupported	B00L IsReBarSupported() const;
ione2aloupponed	Returns TRUE if the rebar can be used.
IskeBarUsed	B00L IsReBarUsed() const;
	Returns TRUE if the frame is using a rebar control.

## Overridables

These functions should not be called directly. The framework will call these functions as required.

CustomDrawToolBar	<pre>virtual LRESULT CustomDrawToolBar(NMHDR* pNMHDR); Draws the ToolBar.</pre>
DrawReBarBkgnd	virtual BOOL DrawReBarBkgnd(CDC* pDC, CReBar* pReBar); Draws the ReBar background when ReBar themes are enabled. Returns TRUE whe background drawing is suppressed.
DrawStatusBarBkgnd	virtual BOOL DrawStatusBarBkgnd(CDC* pDC, CStatusBar* pS Draws the StatusBar background when StatusBar themes are enabled. Returns TRU default background drawing is suppressed.
LoadRegistrySettings	<pre>virtual BOOL LoadRegistrySettings(LPCTSTR szKeyName); Loads settings for the application from the registry. By default the size and position of the frame window, and the view state c StatusBar and ToolBar. Override this function to load other set the registry.</pre>
OnActivate	<pre>virtual void OnActivate(WPARAM wParam, LPARAM lParam); Called when the frame window is activated.</pre>
OnClose	<pre>virtual void OnClose();</pre>

	Called when the frame window is closed.
OnCreate	<pre>virtual void OnCreate();</pre>
OnCreate	Called when the frame window is created. This creates the toolbar and status ba function in the derived class to customise the frame's creation.
OnCustromDraw	<pre>virtual LRESULT OnCustomDraw(LPNMHDR pNMHDR);</pre>
oncustomerati	Called when the frame receives a Custom Draw notification.
OnDestroy	virtual void OnDestroy();
Childestiloy	Called when the frame window is about to be destroyed.
OnDrawItem	virtual LRESULT OnDrawItem(WPARAM wParam, LPARAM lPar
	Called when the frame performs owner-drawing of controls or menus.
OnExitMenuLoop	<pre>virtual void OnExitMenuLoop();</pre>
	Called then a popup menu modal loop has exited.
OnHelp	<pre>virtual LRESULT OnHelp();</pre>
L	Called when the F1 key is pressed. It displays the help dialog.
OnInitMenuPopup	<pre>virtual void OnInitMenuPopup(WPARAM wParam, LPARAM lP</pre>
	Called when a popup menu belonging to the frame is about to be displayed.
o	<pre>virtual void OnInitialUpdate();</pre>
OnInitialUpdate	Override this function to specify what happens after the frame window and all c created.
OnMeasureItem	virtual LRESULT OnMeasureItem(WPARAM wParam, LPARAM l
Omvicusurenem	Called when when the size of an owner drawn control or menu is required.
	virtual LRESULT OnMenuChar(WPARAM wParam, LPARAM lPar
OnMenuChar	Called when a menu mnemonic character that doesn't match any of the predefine the current menu is pressed.
OnMenuSelect	virtual void OnMenuSelect(WPARAM wParam, LPARAM lPara
	Updates the status bar when a menu item is selected.
OnNotify	virtual LRESULT OnNotify(WPARAM wParam, LPARAM lParam
·J	Called when the frame receives a notification by way of a WM_NOTIFY messa

OnSetFocus	<pre>virtual void OnSetFocus();</pre>
	Called when the frame gets keyboard focus.
OnRBNHeightChange	<pre>virtual LRESULT OnRBNHeightChange(LPNMHDR pNMHDR);</pre>
	Called when the height of the ReBar has changed.
OnRBNLayoutChanged	
	Called when the layout of the ReBar has changed.
OnRBNMinMax	<pre>virtual LRESULT OnRBNMinMax(LPNMHDR pNMHDR); Called when a ReBar band is about to be maximized or minimized.</pre>
OnSysColorChange	<pre>virtual void OnSysColorChange();</pre>
, ,	Called when the system colours are changed, perhaps in response to changing th
	<pre>virtual void OnSysCommand();</pre>
OnSysCommand	Called when the user chooses a command from the <i>Window</i> menu (formerly kno or control menu) or when the user chooses the maximize button, minimize butto or close button.
OnTimer	<pre>virtual void OnTimer(WPARAM wParam);</pre>
	Called at regular intervals to update the status bar.
OnTBNDropDown	<pre>virtual LRESULT OnTBNDropDown(LPNMT00LBAR pNMTB);</pre>
-	Called when the user clicks a drop down button on a ToolBar
OnTTNGetDispInfo	<pre>virtual LRESULT OnTTNGetDispInfo(LPNMTTDISPINF0 pNMTD]</pre>
•	Called when a ToolTip notification is recieved.
OnUndocked	<pre>virtual LRESULT OnUndocked();</pre>
	Called when a child window has been undocked.
OnViewStatusBar	<pre>virtual void OnViewStatusBar();</pre>
	Called when the StatusBar is to be displayed or hidden.
OnViewToolBar	<pre>virtual void OnViewToolBar();</pre>
	Called when the toolbar is to be displayed or hidden.
PreCreate	<pre>virtual void PreCreate(CREATESTRUCT&amp; cs);</pre>

	Sets the window parameters specified in CREATESTRUCT prior to window creatic
PreRegisterClass	<pre>virtual void PreRegisterClass(WNDCLASS &amp;wc);</pre>
	Sets the window class parameters specified in WNDCLASS prior to window creation
Durrandat Marsa	<pre>virtual BOOL PreTranslateMessage(MSG* pMsg);</pre>
PreTranslateMessage	Used to translate window messages before they are dispatched to the TranslateMess DispatchMessage Windows functions.
	<pre>virtual B00L SaveRegistrySettings();</pre>
SaveRegistrySettings	Saves settings for the application to the registry. By default this saves the size and p frame window. Override this function to save other settings to the registry.
	<pre>virtual void SetupToolBar();</pre>
SetupToolBar	Override this function to assign resource IDs to toolbar buttons using the AddToolE function. All other toolbar configuration can be specified in this function as well.
	<pre>virtual void SetTheme();</pre>
SetTheme	Sets the theme used when menu items are selected.

# Operations

AddMenuBarBand	<pre>virtual void AddMenuBarBand(); Adds a menubar to the rebar.</pre>
AddMenuIcon	virtual BOOL AddMenuIcon(int nID_MenuItem, HICON hIcon Adds an icon to be used on drop down menu items from the supplied icon.
AddMenuIcons	AddMenuIcons(const std::vector <uint>&amp; MenuData, COLORI Adds a set of icons to be used on drop down menu items from the supplied bitm</uint>
AddMRUEntry	<pre>virtual void AddMRUEntry(LPCTSTR szMRUEntry); Adds the specified text to the Most Recently Used list.</pre>
AddToolBarBand	virtual void AddToolBarBand(CToolBar* pTB, DWORD dwSty Adds a toolbar to the rebar.
AddToolBarButton	virtual void AddToolBarButton(UINT nID, BOOL bEnabled Adds a button to the ToolBar.

AdjustFrameRect	<pre>virtual void AdjustFrameRect(RECT rcView) const;</pre>
5	Adds a resource ID to the set used by the toolbar, and specifies if the buttor
CreateToolBar	<pre>virtual void CreateToolBar();</pre>
	Create's the frame's ToolBar.
DrawMenuItem	<pre>virtual void DrawMenuItem(LPDRAWITEMSTRUCT pdis);</pre>
	Draws the icon on drop down menu items.
DrawMenuItemBkgnd	<pre>virtual void DrawMenuItemBkgnd(LPDRAWITEMSTRUCT pd</pre>
C	Draws the icon on drop down menu items.
DrawMenuItemCheckmark	<pre>virtual void DrawMenuItemCheckmark(LPDRAWITEMSTRUC</pre>
	Draws the check marks on drop down menu items.
DrawMenuItemIcon	virtual void DrawMenuItemIcon(LPDRAWITEMSTRUCT pdi
	Draws the icon on drop down menu items.
DrawMenuItemText	<pre>DrawMenuItemText(LPDRAWITEMSTRUCT pdis);</pre>
	Draws the text on drop down menu items.
LoadRegistryMRUSettings	<pre>virtual BOOL LoadRegistryMRUSettings(UINT nMaxMRU</pre>
0 1 0	Loads a list of Most Recently Used files from the registry.
MeasureMenuItem	<pre>virtual void MeasureMenuItem(MEASUREITEMSTRUCT *pm</pre>
	Calculates the required size of the menu item when performing owner draw
RecalcLayout	<pre>virtual void RecalcLayout();</pre>
c .	Repositions the child windows of the frame, such as the toolbar, status bar
emoveMRUEntry	<pre>virtual void RemoveMRUEntry(LPCTSTR szMRUEntry);</pre>
5	Removes the specified text from the Most Recently Used list.
etMenuBarBandSize	<pre>virtual void SetMenuBarBandSize();</pre>
	Adjusts the size of the MenuBar when the frame is resized.
SetMenuIcons	virtual UINT SetMenuIcons(const std::vector <uint>&amp;</uint>
Setwienurcons	Replaces the set of icons to be used on drop down menu icons from the sup

SetStatusIndicators	<pre>virtual void SetStatusIndicators();</pre>
	Updates the status bar with changes to the CAPS, NUM LOCK, and SCRL indic
SetStatusText	<pre>virtual void SetStatusText();</pre>
	Sets the text of the status bar.
SetTBImageList	<pre>virtual void SetTBImageList(CToolBar* pToolBar, CImage</pre>
occi Dinagellist	Sets the image list for the specified ToolBar.
SetTBImageListDis	virtual void SetTBImageListDis(CToolBar* pToolBar, CI
See Dinage List Dis	Sets the disabled image list for the specified ToolBar.
SetTBImageListHot	<pre>virtual void SetTBImageListHot(CToolBar* pToolBar, CIn</pre>
SetTomageListriot	Sets the hot disabled image list for the specified ToolBar.
SetToolBarImages	virtual void SetToolBarImages(COLORREF crMask, UINT Te
SetToolDarmages	Sets the image lists for the toolbar from the specified bitmap IDs.
Char Manu	<pre>virtual void ShowMenu(BOOL bShow);</pre>
ShowMenu	Show or hide the menu.
ShowStatusBar	<pre>virtual void ShowStatusBar(BOOL bShow);</pre>
	Show or hide the statusbar.
ShowToolBar	virtual void ShowToolBar(BOOL bShow):
_	
UpdateMRUMenu	
	<pre>virtual void ShowStatusBar(BOOL bShow); Show or hide the statusbar. virtual void ShowToolBar(BOOL bShow); Show or hide the toolbar. virtual void UpdateMRUMenu(); Updates the menu to display the MRU entries added by the AddMRUEntry fund</pre>

## **Member Variables**

	DWORD m_ShowCmd;
	The initial show state of the frame, loaded from the registry. It is set to either SW_MAXIMIZE or SW_SHOW.
	BOOL m_UseIndicatorStatus;
m_UseIndicatorStatus	Set to TRUE to see indicators in status bar

m UseMenuStatus	BOOL m_UseMenuStatus;
	Set to TRUE to see menu and toolbar updates in status bar.
m_UseReBar	BOOL m_UseReBar;
	Set to TRUE if ReBars are to be used.
m_UseThemes	BOOL m_UseThemes;
S	Set to TRUE if themes are to be used.
m_UseToolBar	BOOL m_UseToolBar;
	Set to TRUE if the toolbar is used

#### **Base class Members**

For base class members, refer to the members of <u>CDocker</u>.

### Remarks

Refer to the <u>Frames</u> section for information on how to use this class to create frames.

Header file	frame.h
Win32/64 support	Yes
WinCE support	No
Library required	Comctl32.lib

# **CGDIObject Class**

# Description

This is the class that <u>CBitmap</u>, <u>CBrush</u>, <u>CFont</u>, <u>CPalette</u>, <u>CPen</u>, and <u>CRgn</u> inherit from.

# **CGDIObject Members**

### Initialisation and Assignment

CGDIObject ();

Constructor for CGDIObject.

# CGDIObject

```
CGDIObject(const CGDIObject& rhs);
```

Constructor for CGDIObject that creates a copy. CGDIObjects are reference counted, and each copy manages the same underlying HGDIOBJ. Attach

void Attach(HGDI0BJ h0bject);

Attaches a GDI HANDLE to the CGDIObject. The HGDIOBJ will be automatically deleted when the destructor is called unless it is detached. Detach

#### HGDIOBJ Detach();

Detaches the HGDIOBJ from this object. GetHandle

HGDIOBJ GetHandle() const;

Returns the GDI handle (HGDIOBJ) associated with this object. GetObject

int GetObject(int nCount, LPVOID pObject) const;

Retrieves information for the specified graphics object. **Operator** =

void operator = (HGDIOBJ hObject);

Assigns a HGDIOBJ to the CGDIObject. **Operator** =

#### CGDIObject& operator = ( const CGDIObject& rhs );

Creates a copy of the CGDIObject. CGDIObjects are reference counted, and each copy manages the same underlying HGDIOBJ.

## Remarks

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	file.h
Win32/64 support	Yes
WinCE support	Yes

# **CHeader Class**

## Description

The CHeader class adds support for the header control. A header control is a window that is usually positioned above columns of text or numbers. It contains a title for each column, and it can be divided into parts. The user can drag the dividers that separate the parts to set the width of each column.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of animation controls.

## **CHeader Members**

# Construction CHeader CHeader(); Constructs a header control. Attributes GetBitmapMargin int GetBitmapMargin() const; Gets the width of the bitmap margin for a header control. GetImageList HIMAGELIST GetImageList() const; Gets the handle to the image list that has been set for the header control. GetItem BOOL GetItem(int nPos, HDITEM\* pHeaderItem) const; Gets information about an item in a header control. GetItemCount int GetItemCount() const; Gets a count of the items in a header control. GetItemRect CRect GetItemRect(int nIndex) const; Gets the bounding rectangle for a given item in a header control. GetOrderArray

#### B00L GetOrderArray(LPINT piArray, int iCount);

Gets the current left-to-right order of items in a header control. OrderToIndex

#### int OrderToIndex(int nOrder) const;

Retrieves an index value for an item based on its order in the header control. SetBitmapMargin

int SetBitmapMargin(int nWidth);

Sets the width of the margin, specified in pixels, of a bitmap in the header control. SetImageList

HIMAGELIST SetImageList(HIMAGELIST himl);

Assigns an image list to the header control. SetItem

BOOL SetItem(int nPos, HDITEM\* pHeaderItem);

Sets the attributes of the specified item in a header control.. SetOrderArray

B00L SetOrderArray(int iCount, LPINT piArray);

Sets the left-to-right order of header items. **Operations** ClearAllFilters

int ClearAllFilters();

Clears all of the filters for the header control. ClearFilter

int ClearFilter(int nColumn);

Clears the filter for the header control CreateDragImage

HIMAGELIST CreateDragImage(int nIndex);

Creates a transparent version of an item image within the header control. DeleteItem

B00L DeleteItem(int nPos);

Deletes an item from the header control. EditFilter

int EditFilter(int nColumn, BOOL bDiscardChanges);

Moves the input focus to the edit box when a filter button has the focus. InsertItem

int InsertItem(int nPos, HDITEM\* phdi);

Inserts a new item into the header control. Layout

B00L Layout(HDLAYOUT\* pHeaderLayout);

Retrieves the correct size and position of the header control within the parent window.

# SetFilterChangeTimeout

int SetFilterChangeTimeout(DWORD dwTimeOut);

Sets the timeout interval between the time a change takes place in the filter attributes and the posting of an HDN\_FILTERCHANGE notification. SetHotDivider

```
int SetHotDivider(CPoint pt);
```

int SetHotDivider(int nIndex);

Changes the color of a divider between header items to indicate the destination of an external drag-and-drop operation. **Overridables** PreRegisterClass

virtual void PreRegisterClass(WNDCLASS &wc);

Set the window class parameters before the window is created.

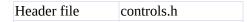
**Base class Members** 

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

A header control can be created as a child window of another control, such as a list box. However, the parent control is not aware of the header control and does not allow for the space taken up by the header, with the result that list items will appear behind the header. If you wish to use a header control in a list box or other control, the parent control must be owner-drawn so that all items are displayed in the correct position.

List view controls already have header controls. Instead of creating a header control for a list view, you use LVM\_GETHEADER or ListView\_GetHeader to retrieve the existing control.



WinCE supportYesLibrary requiredComctl32.lib	Win32/64 support	Yes
Library required Comctl32.lib	WinCE support	Yes
	Library required	Comctl32.lib

# **CHotKey Class**

## Description

The CHotKey class adds support for the hot key control. A hot key control is a window that enables the user to enter a combination of keystrokes to be used as a hot key.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of animation controls.

## **CHotKey Members**

CHotKey ();

Constructs a hot key control.

## GetHotKey

```
DWORD GetHotKey() const;
```

Gets the virtual key code and modifier flags of a hot key from the hot key control. GetKeyName

```
CString GetKeyName(UINT vk, BOOL fExtended) const;
```

Translates the virtual-key code to a scan code. **SetHotKey** 

void SetHotKey(DWORD dwKey);

PSets the hot key combination for a hot key control. SetRules

void SetRules(WORD wInvalidComb, WORD wModifiers);

Defines the invalid combinations and the default modifier combination for a hot key control.

## **Overridables** PreRegisterClass

virtual void PreRegisterClass(WNDCLASS &wc);

Set the window class parameters before the window is created.

**Base class Members** 

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

A hot key is a key combination that the user can press to perform an action quickly. For example, a user can create a hot key that activates a given window and brings it to the top of the z-order. The hot key control displays the user's choices and ensures that the user selects a valid key combination.

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CImageList Class**

## Description

The CImageList class adds support for the image lists. An image list is a collection of images of the same size, each of which can be referred to by its index. Image lists are used to efficiently manage large sets of icons or bitmaps. All images in an image list are contained in a single, wide bitmap in screen device format. An image list can also include a monochrome bitmap that contains masks that are used to draw images transparently.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of image lists.

## **CImageList Members**

```
CImageList();
CImageList
           Constructs a CImageList.
Add
int Add(CBitmap* pbmImage, CBitmap* pbmMask) ;
int Add(CBitmap* pbmImage, COLORREF crMask);
int Add(HICON hIcon);
Adds an image or images to an image list. Attach
void Attach(HIMAGELIST hImageList);
Attaches an existing image list to this CImageList. BeginDrag
int Add(CBitmap* BOOL BeginDrag(int nImage, CPoint ptHotSpot) const;, CB
Begins dragging an image. Create
BOOL Create(int cx, int cy, UINT nFlags, int nInitial, int nGrow);
BOOL Create(UINT nBitmapID, int cx, int nGrow, COLORREF crMask);
BOOL Create(LPCTSTR lpszBitmapID, int cx, int nGrow, COLORREF crMask);
```

#### BOOL Create(CImageList\* pImageList);

Creates the image list and assigns it to the CImageList. CreateDisabledImageList

#### HIMAGELIST CreateDisabledImageList( HIMAGELIST himlNormal );

Creates a gray scale image list from the specified color image list. DeleteImageList

#### void DeleteImageList();

Destroys the image list. Detach

#### HIMAGELIST Detach();

Detaches the HIMAGELIST from this CImageList. If the HIMAGELIST is not detached it will be destroyed when this CImageList is deconstructed. DragEnter

BOOL DragEnter(CWnd\* pWndLock, CPoint point) const;

Displays the drag image at the specified position within the window. DragLeave

B00L DragLeave(CWnd\* pWndLock) const;

Unlocks the specified window and hides the drag image, allowing the window to be updated. DragMove

BOOL DragMove(CPoint pt) const;

Moves the image that is being dragged during a drag-and-drop operation. This function is typically called in response to a WM\_MOUSEMOVE message. DragShowNolock

BOOL DragShowNolock(BOOL bShow) const;

Shows or hides the drag image during a drag operation, without locking the window. Draw

BOOL Draw(CDC\* pDC, int nImage, POINT pt, UINT nStyle) const;

Draws an image list item in the specified device context. DrawEx

BOOL DrawEx(CDC\* pDC, int nImage, POINT pt, SIZE sz, COLORREF clrBk, COL

Draws an image list item in the specified device context. The function uses the specified drawing style and blends the image with the specified color. DrawIndirect

BOOL DrawIndirect(IMAGELISTDRAWPARAMS\* pimldp);

Draws an image list image based on an IMAGELISTDRAWPARAMS structure. From Handle

static CImageList\* FromHandle(HIMAGELIST hImageList);

Returns the CImageList associated with the specified image list handle. If a CImageList object doesn't already exist, a temporary CImageList object is created. This temporary CImageList will be deleted sometime after the processing of the current message is complete.. GetIcon

HICON GetIcon(int iImage, UINT nFlags) const;

Creates an icon from an image and mask in an image list. GetIcon

HICON GetIcon(int iImage, UINT nFlags) const;

Creates an icon from an image and mask in an image list. GetIconSize

B00L GetIconSize(int\* cx, int\* cy) const;

Retrieves the dimensions of images in an image list. GetImageCount

int GetImageCount() const;

Retrieves the number of images in an image list. GetImageInfo

BOOL GetImageInfo(int nImage, IMAGEINFO\* pImageInfo) const;

Retrieves information about an image. Remove

BOOL Remove(int nImage) const;

Removes an image from an image list. **Replace** 

BOOL Replace(int nImage, CBitmap\* pbmImage, CBitmap\* pbmMask) const;

int Replace(int nImage, HICON hIcon) const;

Replaces an image in an image list with a new image. Operator HIMAGELIST

operator HIMAGELIST () const;

Retrieves the image list's handle.

## Remarks

A CImageList object will automatically destroy the image list associated with it

when its destructor is called. Detach the HIMAGELIST if you don't want it destroyed when the CImageList object goes out of scope.

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CIPAddress Class**

## Description

The CIPAddress class adds support for the IP Address control. An Internet Protocol (IP) address control allows the user to enter an IP address in an easily understood format. This control also allows the application to obtain the address in numeric form rather than in text form.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of IP Address controls.

## **CIPAddress Members**

CIPAddress();

Constructs the IP address control.

## ClearAddress

```
void ClearAddress();
```

Clears the contents of the IP address control. GetAddress

```
int GetAddress(BYTE& nField0, BYTE& nField1, BYTE& nField2, BYTE& nField
```

```
int GetAddress(DWORD* dwAddress);
```

Gets the address values for all four fields in the IP address control. Is Blank

B00L IsBlank() const;

Determines if all fields in the IP address control are blank.  ${\displaystyle SetAddress}$ 

```
void SetAddress(BYTE nField0, BYTE nField1, BYTE nField2, BYTE nField3);
```

void SetAddress(DWORD dwAddress);

Sets the address values for all four fields in the IP address control. SetFieldFocus

void SetFieldFocus(WORD nField);

Sets the keyboard focus to the specified field in the IP address control. All of the text in that field will be

# selected. SetFieldRange

void SetFieldRange(int nField, BYTE nLower, BYTE nUpper);

Sets the valid range for the specified field in the IP address control. **Overridables** PreRegisterClass

virtual void PreRegisterClass(WNDCLASS &wc);

Set the window class parameters before the window is created.

**Base class Members** 

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

An IP address control is not an edit control and it will not respond to EM\_ messages. It will, however, send the owner window the following edit control notifications through the WM\_COMMAND message. Note that the IP address control will also send private IPN\_ notifications through the WM\_NOTIFY message.

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CListBox Class**

## Description

The ListBox control displays a list of text or iconic items. One or multiple items can be selected from the list.

CListBox is the class responsible for creating a list-box control. It is typically used in a dialog.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of list box controls.

## **CListBox Members**

#### Construction

CListBox	CListBox();
	Constructor for CListBox.

### Attributes

GetAnchorIndex	<pre>int GetAnchorIndex() const; Returns the index of the item that the mouse last selected.</pre>
GetCaretIndex	<pre>int GetCaretIndex() const; Returns the index of the item that has the focus rectangle.</pre>
GetCount	<pre>int GetCount() const; Returns the number of items in the list box.</pre>
GetCurSel	<pre>int GetCurSel() const; Returns the index of the currently selected item.</pre>
GetHorizontalExtent	<pre>int GetHorizontalExtent() const; Returns the scrollable width, in pixels, of a list box.</pre>

GetItemData	<pre>DWORD GetItemData(int nIndex) const;</pre>
	Returns the value associated with the specified item.
GetItemDataPtr	<pre>void* GetItemDataPtr(int nIndex) const;</pre>
	Returns the value associated with the specified item.
GetItemHeight	<pre>int GetItemHeight(int nIndex) const;</pre>
	Returns the height, in pixels, of an item in a list box.
GetItemRect	<pre>int GetItemRect(int nIndex, LPRECT lpRect) const;</pre>
ottiteiniveet	Retrieves the client coordinates of the specified list box item.
	LCID GetLocale() const;
GetLocale	Retrieves the locale of the list box. The high-order word contains the country/reg code and the low-order word contains the language identifier.
GetSel	<pre>int GetSel(int nIndex) const;</pre>
	Returns the selection state of a list box item.
GetSelCount	<pre>int GetSelCount() const;</pre>
	Returns the number of selected items in a multiple-selection list box.
	<pre>int GetSelItems(int nMaxItems, LPINT rgIndex) const;</pre>
GetSelItems	Creates an array of the indexes of all selected items in a multiple-selection list be and returns the total number of selected items.
GetText	<pre>int GetText(int nIndex, LPTSTR lpszBuffer) const;</pre>
	Retrieves the string associated with a specified item and the length of the string.
GetTextLen	<pre>int GetTextLen(int nIndex) const;</pre>
Gerrexteen	Returns the length, in characters, of the string associated with a specified item.
GetTopIndex	<pre>int GetTopIndex() const;</pre>
Gerropindex	Returns the index of the first visible item in a list box.
SetAnchorIndex	<pre>void SetAnchorIndex(int nIndex) const;</pre>
Jeranchormuex	Sets the item that the mouse last selected to a specified item.
	<pre>int SetCaretIndex(int nIndex, BOOL bScroll) const;</pre>

	Sets the focus rectangle to a specified list box item.	
SetColumnWidth	<pre>void SetColumnWidth(int cxWidth) const; Sets the width, in pixels, of all columns in a list box.</pre>	
SetCurSel	<pre>int SetCurSel(int nSelect) const; Selects a specified list box item.</pre>	
SetHorizontalExtent	<pre>void SetHorizontalExtent(int cxExtent) const; Sets the scrollable width, in pixels, of a list box.</pre>	
SetItemData	<pre>int SetItemData(int nIndex, DWORD dwItemData) const; Associates a value with a list box item.</pre>	
SetItemDataPtr	<pre>int SetItemDataPtr(int nIndex, void* pData) const; Sets the 32-bit value associated with the specified item in a list box to be the specified pointer.</pre>	
SetItemHeight	<pre>int SetItemHeight(int nIndex, UINT cyItemHeight) const; Sets the height, in pixels, of an item or items in a list box.</pre>	
SetLocale	LCID SetLocale(LCID nNewLocale) const; Sets the locale of a list box and returns the previous locale identifier.	
SetSel	<pre>int SetSel(int nIndex, BOOL bSelect) const; Selects an item in a multiple-selection list box.</pre>	
SetTabStops	<pre>SetTabStops(int nTabStops, LPINT rgTabStops) const; BOOL SetTabStops(const int&amp; cxEachStop) const; BOOL SetTabStops() const; Sets the tab stops to those specified in a specified array.</pre>	
SetTopIndex	<pre>int SetTopIndex(int nIndex) const; Scrolls the list box so the specified item is at the top of the visible range.</pre>	

# Operations

int AddString(LPCTSTR lpszItem) const;

AddString	Adds a string to a list box and returns its index.	
DeleteString	<pre>int DeleteString(UINT nIndex) const;</pre>	
5	Removes a string from a list box and returns the number of strings remaining in the list.	
Dir	<pre>int Dir(UINT attr, LPCTSTR lpszWildCard) const;</pre>	
	Adds a list of filenames to a list box and returns the index of the last filename added.	
FindString	<pre>int FindString(int nStartAfter, LPCTSTR lpszItem) const;</pre>	
0	Returns the index of the first string in the list box that begins with a specified string.	
FindStringExact	<pre>int FindStringExact(int nIndexStart, LPCTSTR lpszFind) const;</pre>	
	Returns the index of the string in the list box that is equal to a specified string.	
InsertString	<pre>int InsertString(int nIndex, LPCTSTR lpszItem) const;</pre>	
	Inserts a string at a specified index in a list box.	
ItemFromPoint	UINT ItemFromPoint(CPoint pt, B00L& b0utside ) const;	
	Retrieves the zero-based index of the item nearest the specified point in a list box.	
ResetContent	<pre>void ResetContent() const;</pre>	
	Removes all items from a list box.	
SelectString	<pre>int SelectString(int nStartAfter, LPCTSTR lpszItem) const;</pre>	
0	Selects the first string it finds that matches a specified prefix.	
SelItemRange	<pre>int SelItemRange(BOOL bSelect, int nFirstItem, int nLastItem)</pre>	
	Selects a specified range of items in a list box.	

## Overidables

PreRegisterClass	Virtual void PreRegisterClass(WNDCLASS &wc);
	Set the window class parameters before the window is created.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

# Remarks

Like all common controls, the ListBox control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a ListBox control.

List box items can be represented by text strings, bitmaps, or both. If the list box is not large enough to display all the list box items at once, the list box provides a scroll bar. The user scrolls through the list box items, and applies or removes selection status as necessary. Selecting a list box item changes its visual appearance, usually by changing the text and background colors to those specified by the relevant operating system metrics. When the user selects or deselects an item, the system sends a notification message to the parent window of the list box.

Header file	stdcontrols.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CListView Class**

## Description

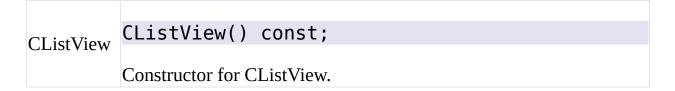
A list-view control is a window that displays a collection of items. Each item consists of an icon and a label. List-view controls provide several ways to arrange and display items. For example, additional information about each item can be displayed in columns to the right of the icon and label.

CListView is the class responsible for creating a list-view control. It can be used as a control in a Dialog, or as the View window in a Frame, Docker, or MDI child.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of list view controls.

## **CListView Members**

#### Construction



### Attributes

ApproximateViewRect	CSize ApproximateViewRect(CSize sz = CSize(-1, -1), in <sup>+</sup> Calculates the approximate width and height required to display a given number c
GetBkColor	COLORREF GetBkColor() const; Retrieves the background color of a list-view control.
GetBkImage	BOOL GetBkImage( LVBKIMAGE& lvbkImage ) const; Retrieves the background image in a list-view control.
GetCallbackMask	UINT GetCallbackMask( ) const; Retrieves the callback mask for a list-view control.
	BOOL GetCheckState( UINT nItem ) const;

GetCheckState	Determines if an item in a list-view control is selected.
GetColumn	BOOL GetColumn( int iCol, LVCOLUMN& Column ) const;
	Retrieves the attributes of a list-view control's column.
GetColumnOrderArray	BOOL GetColumnOrderArray( LPINT piArray, int iCount
	Retrieves the current left-to-right order of columns in a list-view control.
GetColumnWidth	<pre>int GetColumnWidth( int iCol ) const;</pre>
	Retrieves the width of a column in report or list-view.
	<pre>int GetCountPerPage( ) const;</pre>
GetCountPerPage	Calculates the number of items that can fit vertically in the visible area of a lis Only fully visible items are counted.
GetEditControl	HWND GetEditControl( ) const;
	Retrieves the handle to the edit control being used to edit a list-view item's tex
GetExtendedStyle	DWORD GetExtendedStyle( ) const;
	Retrieves the extended styles that are currently in use for a given list-view con
GetHeader	HWND GetHeader( ) const;
	Retrieves the handle to the header control used by a list-view control.
GetHotCursor	HCURSOR GetHotCursor( );
	Retrieves the HCURSOR used when the pointer is over an item while hot trac
GetHotItem	<pre>int GetHotItem( ) const;</pre>
	Retrieves the index of the hot item.
GetHoverTime	DWORD GetHoverTime( ) const;
	Retrieves the amount of time that the mouse cursor must hover over an item b
GetImageList	<pre>CImageList* GetImageList( int nImageType ) const;</pre>
	Retrieves the handle to an image list used for drawing list-view items.
GetItem	BOOL GetItem( LVITEM& lvItem ) const;
	Retrieves some or all of a list-view item's attributes.

GetItemCount	<pre>int GetItemCount( ) const;</pre>
	Retrieves the number of items in a list-view control.
GetItemData	<pre>DWORD_PTR GetItemData( int iItem ) const;</pre>
octitemedua	Retrieves the application data from a list-view's item.
GetItemPosition	BOOL GetItemPosition( int iItem, CPoint& pt ) const;
Settem ostion	Retrieves the position of a list-view item.
GetItemRect	BOOL GetItemRect( int iItem, CRect& rc, UINT nCode )
Gettenikeet	Retrieves the bounding rectangle for all or part of an item in the current view.
GetItemState	<pre>UINT GetItemState( int iItem, UINT nMask ) const;</pre>
Gentemstate	Retrieves the state of a list-view item.
GetItemText	CString GetItemText( int iItem, int iSubItem, UINT n
GentemText	Retrieves the text of a list-view item or subitem.
GetNextItem	<pre>int GetNextItem( int iItem, int iFlags ) const;</pre>
Genveztitelli	Searches for a list-view item that has the specified properties and bears the spe
GetNumberOfWorkArd	uINT GetNumberOfWorkAreas( ) const;
GenvulliberOfWorkAlt	Retrieves the number of working areas in a list-view control.
CotOrigin	B00L GetOrigin( CPoint& pt ) const;
GetOrigin	Retrieves the current view origin for a list-view control.
Cathological	<pre>UINT GetSelectedCount( ) const;</pre>
GetSelectedCount	Determines the number of selected items in a list-view control.
	<pre>int GetSelectionMark( );</pre>
GetSelectionMark	Retrieves the selection mark from a list-view control.
	<pre>int GetStringWidth( LPCTSTR pszString ) const;</pre>
GetStringWidth	Determines the width of a specified string using the specified list-view control'
	BOOL GetSubItemRect( int iItem, int iSubItem, int iCo
GetSubItemRect	Retrieves information about the rectangle that surrounds a subitem in a list-view
	used only on list-view controls that use the LVS_REPORT style.

GetTextBkColor	COLORREF GetTextBkColor( ) const;
	Retrieves the text background color of a list-view control.
GetTextColor	COLORREF GetTextColor( ) const;
	Retrieves the text color of a list-view control.
GetToolTips	CToolTip* GetToolTips( ) const;
Cerroningo	Retrieves the ToolTip control that the list-view control uses to display ToolT
GetTopIndex	<pre>int GetTopIndex( ) const;</pre>
GetTopIndex	Retrieves the index of the topmost visible item when in list or report view.
GetViewRect	B00L GetViewRect( CRect& rc ) const;
	Retrieves the bounding rectangle of all items in the list-view control.
GetWorkAreas	<pre>void GetWorkAreas( int iWorkAreas, LPRECT pRectArra</pre>
Get Work fields	Retrieves the working areas from a list-view control.
SetBkColor	BOOL SetBkColor( COLORREF clrBk );
SeiDkColoi	Sets the background color of a list-view control.
SetBkImage	B00L SetBkImage( LVBKIMAGE& plvbkImage );
SeiDkillage	Sets the background image in a list-view control.
SetCallbackMask	B00L SetCallbackMask( UINT nMask );
SetCallDaCKIVIASK	Changes the callback mask for a list-view control.
SetCheckState	<pre>void SetCheckState( int iItem, BOOL fCheck = TRUE )</pre>
JeiCHeCKJIdle	Used to select or deselect an item in a list-view control.
SetColum-	BOOL SetColumn( int iCol, const LVCOLUMN& pColumn )
SetColumn	Sets the attributes of a list-view column.
	B00L SetColumnOrderArray( int iCount, LPINT piArray
SetColumnOrderArray	Sets the left-to-right order of columns in a list-view control.
	BOOL SetColumnWidth( int iCol, int cx );
SetColumnWidth	Used to change the width of a column in report view or the width of all colu
	0

SetExtendedStyle	<pre>DWORD SetExtendedStyle( DWORD dwNewStyle );</pre>
	Sets extended styles for list-view controls.
SetHotCursor	HCURSOR SetHotCursor( HCURSOR hCursor );
	Sets the HCURSOR that the list-view control uses when the pointer is over an iter check whether or not hot tracking is enabled, call SystemParametersInfo.
SetHotItem	<pre>int SetHotItem( int nIndex );</pre>
	Sets the hot item in a list-view control.
SetHoverTime	<pre>DWORD SetHoverTime( DWORD dwHoverTime = (DWORD)-1 );</pre>
	Sets the amount of time that the mouse cursor must hover over an item before it is
	CSize SetIconSpacing( int cx, int cy ) const;
SetIconSpacing	CSize SetIconSpacing( CSize sz ) const;
	Sets the spacing between icons in list-view controls set to the LVS_ICON style.
SetImageList	<pre>CImageList* SetImageList( CImageList* pNew, int iImagel</pre>
e contrage	Assigns an image list to a list-view control, and returns the old image list (if any).
	BOOL SetItem( LVITEM& pItem ) const;
SetItem	BOOL SetItem( int iItem, int iSubItem, UINT nMask, LPC UINT nState, UINT nStateMask, LPARAM lPa
	Sets some or all of a list-view item's attributes.
SetItemCount	<pre>void SetItemCount( int iCount ) const;</pre>
bettemobult	Causes the list-view control to allocate memory for the specified number of items
SetItemCountEx	<pre>void SetItemCountEx( int iCount, DWORD dwFlags = LVSICH</pre>
SettemCountEx	Sets the virtual number of items in a virtual list view.
SetItemData	BOOL SetItemData( int iItem, DWORD_PTR dwData ) const;
SettemDutu	Sets the application data for a list-view's item.
SetItemPosition	BOOL SetItemPosition( int iItem, CPoint& pt ) const;
	Moves an item to a specified position in a list-view control (in icon or small icon
	BOOL SetItemState( int iItem, LVITEM& Item ) const;
SetItemState	void SetItemState( int iItem, UINT nState, UINT nMask )

	Changes the state of an item in a list-view control.
SetItemText	void SetItemText( int iItem, int iSubItem, LPCTSTR psz <sup>-</sup> Changes the text of a list-view item or subitem.
SetSelectionMark	<pre>int SetSelectionMark( int iIndex ) const; Sets the selection mark in a list-view control.</pre>
SetTextBkColor	BOOL SetTextBkColor( COLORREF clrBkText ) const; Sets the background color of text in a list-view control.
SetTextColor	BOOL SetTextColor( COLORREF clrText ) const; Sets the text color of a list-view control.
SetToolTips	CToolTip* SetToolTips ( CToolTip* pToolTip ) const; Sets the ToolTip control that the list-view control will use to display ToolTips.
SetWorkAreas	<pre>void SetWorkAreas( int nWorkAreas, CRect&amp; pRectArray ) Sets the working area within a list-view control.</pre>
SubItemHitTest	<pre>int SubItemHitTest( LVHITTESTINF0&amp; htInfo ) const; Determines which list-view item or subitem is located at a given position.</pre>

# Operations

Arrange	BOOL Arrange( UINT nCode ) const;
	Arranges items in icon view.
CreateDragImage	<pre>CImageList* CreateDragImage( int iItem, CPoint&amp; pt ) const;</pre>
	Creates a drag image list for the specified item.
DeleteAllItems	BOOL DeleteAllItems( ) const;
Deleter mitchis	Removes all items from a list-view control.
DeleteColumn	BOOL DeleteColumn( int iCol ) const;
	Removes a column from a list-view control.
	BOOL DeleteItem( int iItem ) const;

DeleteItem	Removes an item from a list-view control.
EditLabel	HWnd EditLabel( int iItem ) const;
	Begins in-place editing of the specified list-view item's text. This implicitly selects and
EnsureVisible	BOOL EnsureVisible( int iItem, BOOL fPartialOK ) const;
	Ensures that a list-view item is either entirely or partially visible, scrolling the list-view
FindItem	<pre>int FindItem( LVFINDINF0&amp; FindInfo, int iStart = -1 ) cons</pre>
	Searches for a list-view item with the specified characteristics.
	<pre>int HitTest( LVHITTESTINF0&amp; HitTestInfo ) const;</pre>
HitTest	<pre>int HitTest( CPoint pt, UINT* pFlags = NULL ) const;</pre>
	Determines which list-view item, if any, is at a specified position.
	int InsertColumn( int iCol, const LVCOLUMN& pColumn ) cons
InsertColumn	<pre>int InsertColumn( int iCol, LPCTSTR pszColumnHeading, int</pre>
	Inserts a new column in a list-view control.
	<pre>int InsertItem( const LVITEM&amp; pItem ) const;</pre>
InsertItem	<pre>int InsertItem( int iItem, LPCTSTR pszText ) const;</pre>
	<pre>int InsertItem( int iItem, LPCTSTR pszText, int iImage ) c</pre>
	Inserts a new item in a list-view control.
RedrawItems	BOOL RedrawItems( int iFirst, int iLast ) const;
	Forces a list-view control to redraw a range of items.
Scroll	BOOL Scroll( CSize sz ) const;
	Scrolls the content of a list-view control.
SortItems	BOOL SortItems( PFNLVCOMPARE pfnCompare, DWORD_PTR dwData
	Uses an application-defined comparison function to sort the items of a list-view control to reflect the new sequence.
	BOOL Update( int iItem ) const;
Update	Updates a list-view item. If the list-view control has the LVS_AUTOARRANGE style,

### Overidables

PreRegisterClass Virtual void PreRegisterClass(WNDCLASS &wc); Set the window class parameters before the window is created.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Like all common controls, the list-view control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a list-view control.

Refer to the explorer sample to see a demonstration of the <u>CListView</u> and <u>CTreeView</u> classes.

Header file	listview.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CMDIChild Class**

## Description

This class provides a child window for MDI (Multi Document Interface) applications. MDI applications are capable of hosting several child windows within a frame.

The CMDIChild class inherits much of its default implementation from CWnd, and adds the additional functionality required for MDI applications.

## **CMDIChild Members**

CMDIChild();			
Constructor for CMDIChild.			
GetMDIFrame			
CMDIFrame* GetMDIFrame() const;			
Returns a pointer to the MDI Frame. ${\operatorname{GetView}}$			
<pre>virtual CWnd* GetView() const {return m_pView;}</pre>			
Returns a pointer to the MDI Child's view window.			
MDIActivate			
<pre>void MDIActivate() const;</pre>			
Activates this MDI child. MDIDestroy			
<pre>void MDIDestroy() const;</pre>			
Destroys this MDI child. MDIMaximize			
void MDIMaximize() const;			
Maximizes this MDI Child. MDIRestore			
void MDIRestore() const;			

Restores this MDI child from a maximized or minimized state. RecalcLayout

#### virtual void RecalcLayout();

Repositions the view window when the MDI child is restored. SetChildMenu

virtual void SetChildMenu(LPCTSTR MenuName);

```
Sets the MDI child menu, given its Resource ID. SetHandles
```

void SetHandles(HMENU MenuName, HACCEL AccelName);

Sets the MDI child's menu and accelerator table. SetView

```
virtual void SetView(CWnd& pwndView);
```

Sets the view window for the MDI child.

### **Overridables**

Create	<pre>virtual HWND Create(HWND hWndParent = NULL); Creates the MDI child window.</pre>
OnClose	<pre>virtual void OnClose();</pre>
	Called when the MDI child window is about to be closed. Override this to prevent the M window from being closed.
OnCreate	<pre>virtual void OnCreate();</pre>
Olicieate	Called when the MDI child window is created.
OnMDIActivate	virtual LRESULT OnMDIActivate(WPARAM wParam, LPARAM lParam)
	Called when the MDI child window is activated.
OnWindowPosChanged	virtual LRESULT OnWindowPosChanged(WPARAM wParam, LPARAM lF
	Called when the MDI child window is moved.

#### **Base class Members**

For base class members, refer to the members of CWnd.

## Remarks

### Setting the MDI Child's View window and Menu

Each type of MDI child will have a view window and will usually have a unique menu. The menu which is displayed in the MDI frame will belong the the MDI child that has keyboard focus, or the MDI frame's menu if no MDI child is active. The SetView function sets the view window, and the SetChildMenu seta the menu of a MDI child, as follows.

```
CSimpleMDIChild::CSimpleMDIChild()
{
    // Set m_View as the view window of the MDI child
    SetView(m_View);
    // Set the menu for this MDI child
    SetChildMenu(_T("MdiMenuView"));
}
```

The MDI child's menu would be defined in the resource script file (usually Resource.rc) as follows.

```
MDIMENU MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "New &View\tCtrl+N",
                                                 IDM FILE NEW
        MENUITEM "&Close",
                                                 IDM FILE_CLOSE
        MENUITEM SEPARATOR
                                                 IDM_FILE EXIT
        MENUITEM "E&xit",
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Tool Bar",
                                                 IDW VIEW TOOLBAR, CHECKE
                                                 IDW_VIEW_STATUSBAR, CHEC
        MENUITEM "&Status Bar",
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&Cascade\tShift+F5",
                                                 IDW WINDOW CASCADE
        MENUITEM "&Tile\tShift+F4",
                                                 IDW WINDOW TILE
        MENUITEM "Arrange & Icons",
                                                 IDW WINDOW ARRANGE
        MENUITEM "Close &All",
                                                 IDW WINDOW CLOSEALL
    END
    POPUP "&Help"
    BEGIN
```

```
MENUITEM "&About",
END
END
```

IDM\_HELP\_ABOUT

### Setting the MDI Child's Title and Icon

Each type of MDI child usually has an icon. The MDI child's icon and title are is set as follows.

```
void CMDIChildView::OnInitialUpdate()
{
    ::SetWindowText(m_hWnd, _T("MDI Child Window"));
    SetIconLarge(IDI_VIEW);
    SetIconSmall(IDI_VIEW);
}
```

Refer to the <u>MDI Frames</u> section for information on how to use CMDIChild with <u>CMDIFrame</u>.

Header file	mdi.h
Win32/64 support	Yes
WinCE support	No

# **CMDIFrame Class**

## Description

This class provides a frame for MDI (Multi Document Interface) applications. MDI applications are capable of hosting several child windows within a frame.

The CMDIFrame class inherits much of its default implementation from <u>CFrame</u>, and adds the additional functionality required for MDI frames. The CWnd classes for the MDI child windows should be inherited from <u>CMDIChild</u>.

## **CMDIFrame Members**

#### Constructor



#### Attributes

GetActiveMDIChild	CMDIChild*	<pre>GetActiveMDIChild()</pre>	const;
	Returns the CMDIChild pointer to the active MDI child.		

## GetActiveMenu

#### CMenu\* GetActiveMenu() const;

Returns a pointer to the menu of the active MDI child, or the MDI Frame's menu if no MDI child is active. **GetAllMDIChildren** 

std::vector <MDIChildPtr>& GetAllMDIChildren();

Returns the vector containing the MDI child handles. GetMDIClient

virtual CMDIClient\* GetMDIClient() const;

Returns a pointer to the MDI client window. SetActiveMDIChild

## void SetActiveMDIChild(CMDIChild\* pChild);

Activates the specified MDI child.

### **State functions**

IsMDIChildMa	Returns TRUE if the MDI child is maximised.
IsMDIFrame	virtual BOOL IsMDIFrame() const;
	Overrides CFrame::IsMDIFrame, and returns TRUE.

## Operations

AddMDIChild	<pre>virtual CMDIChild* AddMDIChild(MDIChildPtr pMDIChild);</pre>
	Adds a MDI child window.
MDICascade	<pre>void MDICascade(int nType = 0) const;</pre>
	Arrange all MDI child windows in a cascade format.
MDIIconArrange	<pre>void MDIIconArrange() const;</pre>
	Arrange all minimized MDI child windows.
	<pre>void MDIMaximize() const;</pre>
MDIMaximize	
	Maximize a MDI child window.
MDINext	<pre>void MDINext() const;</pre>
	Activate the next MDI child.
	<pre>void MDIPrev() const;</pre>
MDIPrev	Activate the previous MDI child.
MDIRestore	<pre>void MDIRestore() const;</pre>
	Restores a MDI child window from maximized or minimized size.
MDITile	<pre>void MDITile(int nType = 0) const;</pre>
	Arrange all MDI child windows in a tile format.

RemoveAllMDIChildren	<pre>virtual BOOL RemoveAllMDIChildren();</pre>
	Removes all MDI child windows.
RemoveMDIChild	<pre>virtual void RemoveMDIChild(HWND hWnd);</pre>
KeniovewiDiChild	Removes a MDI child, given its window handle.

### **Overridables**

OnClose	<pre>virtual void OnClose(); Called when the frame window is closed.</pre>
OnViewStatusBar	<pre>virtual void OnViewStatusBar(); Called when the StatusBar is shown.</pre>
OnViewToolBar	<pre>virtual void OnViewToolBar(); Called when the ToolBar window is shown.</pre>
OnWindowPosChanged	<pre>virtual void OnWindowPosChanged(); Called when the size, position, or Z-order of the frame window has changed</pre>
PreTranslateMessage	<pre>virtual BOOL PreTranslateMessage(MSG* pMsg); Used to translate window messages before they are dispatched to the TranslateMessage and DispatchMessage Windows functions.</pre>

#### **Base class Members**

For base class members, refer to the members of CFrame.

## Remarks

Refer to the <u>MDI Frames</u> section for information on how to create MDI Frames and MDI Children.

Header file	mdi.h
Win32/64 support	Yes

WinCE support	No
Library required	Comctl32.lib

# **CMemDC** Class

## Description

The class responsible for creating a memory device context.

# **CMemDC Members**

**Initialization and Assignment** 

Initialization and Assignment	
CMemDC	CMemDC(const CDC* pDC);
	Constructs a CMemDC object.

#### **Base class Members**

For base class members, refer to the members of <u>CDC</u>.

## Remarks

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

# **CMenu Class**

## Description

CMenu adds support for menus to Win32++. A menu is a list of items that specify options or groups of options (a submenu) for an application. Clicking a menu item opens a submenu or causes the application to carry out a command.

A menu is arranged in a hierarchy. At the top level of the hierarchy is the menu bar which contains a list of menus, which in turn can contain submenus. A menu bar is sometimes called a top-level menu, and the and submenus are also known as pop-up menus.

A menu item can either carry out a command or open a submenu. An item that carries out a command is called a command item or a command.

## **CMenu Members**

Initialisation and Assignment	
	CMenu();
CMenu	CMenu(UINT nID);
	Constructor for CMenu.
Attach	

#### Attach

void Attach(HMENU hMenu);

Attaches an existing menu to this CMenu. CreateMenu

```
void CreateMenu();
```

Creates an empty menu. CreatePopupMenu

void CreatePopupMenu();

Creates a drop-down menu, submenu, or shortcut menu. The menu is initially empty..

### DestroyMenu

```
void DestroyMenu();
```

Destroys the menu. Detach

#### HMENU Detach();

Detaches the HMENU from this CMenu. If the HMENU is not detached it will be destroyed when this CMenu is deconstructed. **FromHandle** 

static CMenu\* FromHandle(HMENU hMenu);

Returns the CMenu associated with the specified menu handle. If a CMenu object doesn't already exist, a temporary CMenu object is created. This temporary CMenu will be deleted sometime after the processing of the current message is complete. GetHandle

HMENU GetHandle() const;

Returns the HMENU assigned to this CMenu. LoadMenu

BOOL LoadMenu(LPCTSTR lpszResourceName);

B00L LoadMenu(UINT uIDResource);

BOOL LoadMenuIndirect(const void\* lpMenuTemplate);

Loads the menu from the specified windows resource or template. TrackPopupMenu

BOOL TrackPopupMenu(UINT uFlags, int x, int y, CWnd\* pWnd, LPCRECT lpRec

Displays a shortcut menu at the specified location and tracks the selection of items on the menu.. TrackPopupMenuEx

BOOL TrackPopupMenuEx(UINT uFlags, int x, int y, CWnd\* pWnd, LPTPMPARAMS

Displays a shortcut menu at the specified location and tracks the selection of items on the shortcut menu.

### Menu Item Operations AppendMenu

BOOL AppendMenu(UINT uFlags, UINT\_PTR uIDNewItem = 0, LPCTSTR lpszNewIte

BOOL AppendMenu(UINT uFlags, UINT\_PTR uIDNewItem, const CBitmap\* pBmp);

Appends a new item to the end of the specified menu bar, drop-down menu, submenu, or shortcut menu. CheckMenuItem

UINT CheckMenuItem(UINT uIDCheckItem, UINT uCheck);

Sets the state of the specified menu item's check-mark attribute to either selected or clear. CheckMenuRadioItem

BOOL CheckMenuRadioItem(UINT uIDFirst, UINT uIDLast, UINT uIDItem, UINT

Checks a specified menu item and makes it a radio item. At the same time, the function clears all other

menu items in the associated group and clears the radio-item type flag for those items. DeleteMenu

BOOL DeleteMenu(UINT uPosition, UINT uFlags);

Deletes an item from the specified menu. EnableMenuItem

UINT EnableMenuItem(UINT uIDEnableItem, UINT uEnable);

Enables, disables, or grays the specified menu item. The uEnable parameter must be a combination of either MF\_BYCOMMAND or MF\_BYPOSITION and MF\_ENABLED, MF\_DISABLED, or MF\_GRAYED. GetDefaultItem

UINT GetDefaultItem(UINT gmdiFlags, BOOL fByPos = FALSE);

Determines the default menu item. The gmdiFlags parameter specifies how the function searches for menu items. This parameter can be zero or more of the following values: GMDI\_GOINTOPOPUPS; GMDI\_USEDISABLED. GetMenuContextHelpId

DWORD GetMenuContextHelpId() const;

Retrieves the Help context identifier associated with the menu. GetMenuInfo

BOOL GetMenuInfo(LPMENUINFO lpcmi) const;

Retrieves the menu information. GetMenuItemCount

UINT GetMenuItemCount() const;

Retrieves the number of menu items. GetMenuItemID

UINT GetMenuItemID(int nPos) const;

Retrieves the menu item identifier of a menu item located at the specified position GetMenuItemInfo

BOOL GetMenuItemInfo(UINT uItem, LPMENUITEMINFO lpMenuItemInfo, BOOL fBy

Retrieves information about the specified menu item. GetMenuState

UINT GetMenuState(UINT uID, UINT uFlags) const;

Retrieves the menu flags associated with the specified menu item. Possible values for uFlags are: MF\_BYCOMMAND (default) or MF\_BYPOSITION. GetMenuString

int GetMenuString(UINT uIDItem, LPTSTR lpString, int nMaxCount, UINT uFl

int GetMenuString(UINT uIDItem, CString& rString, UINT uFlags) const;

Copies the text string of the specified menu item into the specified buffer. GetSubMenu

CMenu\* GetSubMenu(int nPos);

Retrieves the CMenu object of a pop-up menu. **InsertMenu** 

BOOL InsertMenu(UINT uPosition, UINT uFlags, UINT\_PTR uIDNewItem = 0, LP BOOL InsertMenu(UINT uPosition, UINT uFlags, UINT\_PTR uIDNewItem, const Inserts a new menu item into a menu, moving other items down the menu. InsertMenuItem

BOOL InsertMenuItem(UINT uItem, LPMENUITEMINFO lpMenuItemInfo, BOOL fByP

Inserts a new menu item at the specified position in a menu. ModifyMenu

BOOL ModifyMenu(UINT uPosition, UINT uFlags, UINT\_PTR uIDNewItem = 0, LP

BOOL ModifyMenu(UINT uPosition, UINT uFlags, UINT\_PTR uIDNewItem, const

Changes an existing menu item. This function is used to specify the content, appearance, and behavior of the menu item. **RemoveMenu** 

BOOL RemoveMenu(UINT uPosition, UINT uFlags);

Deletes a menu item or detaches a submenu from the menu. SetDefaultItem

BOOL SetDefaultItem(UINT uItem, BOOL fByPos = FALSE);

Sets the default menu item for the menu. SetMenuContextHelpId

B00L SetMenuContextHelpId(DWORD dwContextHelpId);

Associates a Help context identifier with the menu. SetMenuInfo

B00L SetMenuInfo(LPCMENUINF0 lpcmi);

Writes the specified buffer to the file. SetMenuItemBitmaps

BOOL SetMenuItemBitmaps(UINT uPosition, UINT uFlags, const CBitmap\* pBmp

Associates the specified bitmap with a menu item. SetMenuItemInfo

BOOL SetMenuItemInfo(UINT uItem, LPMENUITEMINFO lpMenuItemInfo, BOOL fBy

Changes information about a menu item. **Operators** operator HMENU ()

operator HMENU () const;

Retrieves the menu's handle.

## Remarks

CMenu objects can be used anywhere a a handle to a menu (HMENU) might be used. They can be substituted for the HMENU in any of the Windows API functions which use a HMENU as a function argument. The benefit of using a CMenu object is that it automatically deletes the menu when it is destroyed.

Header file	file.h
Win32/64 support	Yes
WinCE support	Yes

# **CMetaFileDC** Class

## Description

The class responsible for creating a metafile device context.

A metafile, also called a vector image, is an image that is stored as a sequence of drawing commands and settings. The commands and settings recorded in a Metafile object can be stored in memory or saved to a file or stream.

## **CMetaFileDC Members**

#### **Initialization and Assignment**

Initialization and Assignment		
CMetaFileDC	<pre>CMetaFileDC();</pre>	
	Constructs a CMetaFileDC object.	
<b>Operations</b> Create		
void Create(L	PCTSTR lpszFilename = NULL);	
Creates a metafile a	and attaches it to the device context. CreateEnhanced	
void CreateEn	<pre>hanced(CDC* pDCRef, LPCTSTR lpszFileName, LPCRECT lpBounds,</pre>	
Creates an enhanced metafile and attaches it to the device context. $\operatorname{Close}$		
HMETAFILE Close();		
Closes the metafile. CloseEnhanced		
HENHMETAFILE CloseEnhanced();		
Closes the enhanced metafile.		
Base class Members		

For base class members, refer to the members of <u>CDC</u>.

# Remarks

To use the CMetaFileDC, create the CMetafileDC object, and then create the metafile or enhanced metafile. Enhanced metafiles are supported by all 32-bit and 64-bit Windows operating systems. These should be the default metafiles used by all modern windows applications.

Once the Metafile is closed the HMETAFILE or HENHMETADILE handle can be used to play the metafile. The handle returned by Close or CloesEnhanced should be deleted when it no longer required.

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

# **CMonthCalendar Class**

## Description

The CMonthCalendar class adds support for the month calendar control. The month calendar control provides the user with an intuitive and recognizable method of entering or selecting a date. The control also provides the application with the means to obtain and set the date information in the control using existing data types.

## **CMonthCalendar Members**

#### Construction

CMonthCalendar ();

Constructs a month calendar control.

#### Attributes GetColor

COLORREF GetColor(int nRegion) const;

Retrieves the color for a given portion of the month calendar control. GetFirstDayOfWeek

int GetFirstDayOfWeek(BOOL\* pbLocal = NULL) const;

Retrieves the first day of the week for the month calendar control. GetMinReqRect

CRect GetMinReqRect() const;

Retrieves the minimum size required to display a full month in the month calendar control. GetMonthDelta

int GetMonthDelta() const;

Retrieves the scroll rate for the month calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. SetColor

#### COLORREF SetColor(int nRegion, COLORREF ref);

Sets the color for a given portion of the month calendar control. SetFirstDayOfWeek

#### B00L SetFirstDayOfWeek(int iDay, int\* lpn0ld = NULL);

Sets the first day of the week for the month calendar control. SetMonthDelta

#### int SetMonthDelta(int iDelta);

Sets the scroll rate for the month calendar control. The scroll rate is the number of months that the control moves its display when the user clicks a scroll button. Operations GetCurSel

BOOL GetCurSel(LPSYSTEMTIME pDateTime) const;

Retrieves the currently selected date. GetMaxSelCount

#### int GetMaxSelCount() const;

Retrieves the maximum date range that can be selected in a month calendar control.

## GetMonthRange

int GetMonthRange(LPSYSTEMTIME pMinRange, LPSYSTEMTIME pMaxRange, DWORD

Retrieves date information (using SYSTEMTIME structures) that represents the high and low limits of the month calendar control's display. GetSelRange

DWORD GetRange(LPSYSTEMTIME pMinRange, LPSYSTEMTIME pMaxRange) const;

Retrieves date information that represents the upper and lower limits of the date range currently selected by the user. GetToday

#### BOOL GetToday(LPSYSTEMTIME pDateTime) const;

Retrieves the date information for the date specified as "today" for the month calendar control. HitTest

DWORD HitTest(PMCHITTESTINF0 pMCHitTest);

Determines which portion of the month calendar control is at a given point on the screen. SetCurSel

#### BOOL SetCurSel(const LPSYSTEMTIME pDateTime);

Sets the currently selected date for the month calendar control. If the specified date is not in view, the control updates the display to bring it into view. SetDayState

#### BOOL SetDayState(int nMonths, LPMONTHDAYSTATE pStates);

Sets the day states for all months that are currently visible within the month calendar control. SetMaxSelCount

B00L SetMaxSelCount(int nMax);

Sets the maximum number of days that can be selected in the month calendar control. SetRange

BOOL SetRange(const LPSYSTEMTIME pMinRange, const LPSYSTEMTIME pMaxRange

Sets the minimum and maximum allowable dates for the month calendar control. SetSelRange

BOOL SetSelRange(const LPSYSTEMTIME pMinRange, const LPSYSTEMTIME pMaxRa

Sets the selection for a month calendar control to the given date range. SetToday

void SetToday(const LPSYSTEMTIME pDateTime);

Sets the "today" selection for a month calendar control.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

### Remarks

Header file	controls.h
Win32/64 support	Yes
support	
WinCE support	Yes

# **CPaintDC Class**

# Description

This class is responsible for creating a device context for drawing to a window during the handling of the WM\_PAINT message. This device context is used by CWnd::OnDraw.

# **CPaintDC Members**

#### Initialization and Assignment

Initialization and Assignment			
CPaintDC	CPaintDC(const CWnd* pWnd);		
	Constructs a CPaintDC object.		

#### **Base class Members**

For base class members, refer to the members of <u>CDC</u>.

### Remarks

This class automatically calls CWnd::BeginPaint when it is constructed, and automatically calls CWnd::EndPaint when it is destroyed.

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

# **CPalette Class**

## Description

The class responsible for creating and managing palette resources.

# **CPalette Members**

**Initialization and Assignment** 

Initializati	on and Assignment
CPalette	CPalette();
Cralette	CPalette(HPALETTE hPalette); Constructs a CPaletteBrush object.
FromHar	ndle

```
static CPalette* FromHandle(HPALETTE hPalette);
```

Returns the CPalette associated with the specified palette handle. If a CPalette object doesn't already exist, a temporary CPalette object is created. This temporary CPalette will be deleted sometime after the processing

of the current message is complete. **Operator HPALETTE()** 

```
operator HPALETTE() const;
```

Allows a CPalette object to be used as a palette handle (HPALETTE). Attributes GetEntryCount

int GetEntryCount() const;

Retrieve the number of entries in the palette. GetNearestPaletteIndex

UINT GetNearestPaletteIndex (COLORREF crColor) const;

Retrieves the index for the entry in the palette most closely matching a specified color value. GetPaletteEntries

UINT GetPaletteEntries(UINT nStartIndex, UINT nNumEntries,

#### LPPALETTEENTRY lpPaletteColors) const;

#### Retrieves a specified range of palette entries from the palette. SetPaletteEntries

#### UINT SetPaletteEntries(UINT nStartIndex, UINT nNumEntries, LPPALETTEENTRY lpPaletteColors);

Sets RGB (red, green, blue) color values and flags in a range of entries in the palette. **Operations** AnimatePalette

#### 

Replaces entries in the palette. CreateHalftonePalette

HPALETTE CreateHalftonePalette(CDC\* pDC);

Creates a halftone palette for the specified device context (DC). CreatePalette

HPALETTE CreatePalette(LPLOGPALETTE lpLogPalette);

Creates a logical palette from the information in the specified LOGPALETTE structure.

#### **Base class Members**

For base class members, refer to the members of CGDIObject.

### Remarks

CPalette objects can be used anywhere a a handle to a palette (HPALETTE) might be used. They can be substituted for the HPALETTE in any of the Windows API functions which use a HPALETTE as a function argument. The benefit of using a CPalette object is that it automatically deletes the brush when it is destroyed.

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

# **CPen Class**

## Description

The class responsible for creating and managing pen resources.

# **CPen Members**

#### Initialization and Assignment

CPen();

CPen(HPEN hPen);

CPen CPen(int nPenStyle, int nWidth, COLORREF crColor);

CPen(int nPenStyle, int nWidth, const LOGBRUSH\* pL int nStyleCount = 0, const DWOR

Constructs a CPen object.

## FromHandle

static CPen\* FromHandle(HPEN hPen);

Returns the CPen associated with the specified pen handle. If a CPen object doesn't already exist, a temporary CPen object is created. This temporary CPen will be deleted sometime after the processing of the current message is complete. **Operator HPEN()** 

operator HPEN() const;

Allows a CPen object to be used as a pen handle (HPEN). Attributes GetExtLogPen

EXTLOGPEN GetExtLogPen() const;

Retrieves the EXTLOGPEN struct that specifies the pen's style, width, color and brush attributes. GetLogPen

```
LOGPEN GetLogPen() const;
```

# Retrieves the LOGPEN struct that specifies the pen's style, width, and color. Operations CreatePen

HPEN CreatePen(int nPenStyle, int nWidth, COLORREF crColor);

Creates a logical pen that has the specified style, width, and color. CreatePenIndirect

#### HPEN CreatePenIndirect(LPL0GPEN lpLogPen);

Creates a logical pen that has the style, width, and color specified in a structure. ExtCreatePen

#### 

Creates a logical cosmetic or geometric pen that has the specified style, width, and brush attributes.

#### **Base class Members**

For base class members, refer to the members of CGDIObject.

### Remarks

CPen objects can be used anywhere a a handle to a pen (HPEN) might be used. They can be substituted for the HPEN in any of the Windows API functions which use a HPEN as a function argument. The benefit of using a CPen object is that it automatically deletes the pen when it is destroyed.

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

# **CPoint Class**

# Description

A CPoint can be used anywhere that a POINT structure could be used.

# **CPoint Members**

CPoint	CPoint();
	CPoint(int X, int Y);
	CPoint(SIZE sz);
	CPoint(POINT pt);
	CPoint(DWORD dw);
	Various methods of constructing a CPoint.

# operator LPPOINT()

operator LPPOINT();

Returns a pointer to the POINT associated with this object. Operator ==

BOOL operator == (POINT pt) const;

Returns TRUE if the co-ordinates of the source point and the CPoint are equal. Operator !=

BOOL operator != (POINT pt) const;

Returns TRUE if the of the source point and the CPoint are not equal. **Operator** +=

void operator += (SIZE sz);

void operator += (POINT pt);

Adds the specified point. **Operator** -=

void operator -= (SIZE sz);

void operator -= (POINT pt);

Subtracts the specified point. **OPERATOR** -

CPoint operator - () const;

Returns the unary minus (additive inverse). **Operator** +

CPoint operator + (SIZE sz) const;

CPoint operator + (POINT pt) const;

CRect operator + (LPCRECT prc) const;

Adds the point and returns the value. **Operator** -

CPoint operator - (SIZE sz) const;

CPoint operator - (POINT pt) const;

CRect operator - (LPCRECT prc) const;

Subtracts the point and returns the value. Offset

void Offset(int dx, int dy);

void Offset(POINT pt);

void Offset(SIZE sz);

Moves the CPoint by the specified offsets. SetPoint

void SetPoint(int X, int Y);

Sets the coordinates of the CPoint.

### Remarks

By default, the constructor sets all the size coordinates zero, but the CPoint can also be constructed from two integers, a size, a point, or a dword.

The CPoint class inherits from the POINT structure. As a result, the data members of the underlying POINT struct are also accessible as data members.

Win32/64 support Yes
11
WinCE support Yes

# **CProgressBar Class**

## Description

The CProgressBar class is used to create and manage a progress bar control. A progress bar is a window that an application can use to indicate the progress of a lengthy operation. It consists of a rectangle that is gradually filled with the system highlight color as an operation progresses.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of progress bar controls.

## **CProgressBar Members**

	CProgressBar();
	Constructor for the CProgressBar.
GetPos	

```
int GetPos() const;
```

Retrieves the current position of the progress bar. GetRange

int GetRange(BOOL fWhichLimit, PPBRANGE ppBRange) const;

Retrieves the information about the current high and low limits of the progress bar. OffsetPos

int OffsetPos(int nIncrement) const;

Advances the current position of a progress bar bt the specified increment and redraws the bar to reflect the new position. **SetPos** 

int SetPos(int nNewPos) const;

Sets the current position of the progress bar and redraws the bar to reflect the new position. SetRange

int SetRange(short nMinRange, short nMaxRange) const;

Sets the minimum and maximum values for the progress bar and redraws the bar to reflect the new range. SetStep

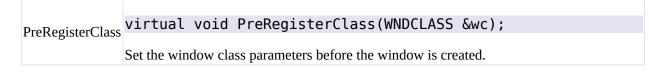
#### int SetStep(int nStepInc) const;

Specifies the step increment for the progress bar, used by StepIt. The default value is 10. StepIt

```
int StepIt() const;
```

Advances the current position of the progress bar by the step increment and redraws the progress bar to reflect the new position.

#### **Overridables**



#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Like all common controls, the progress bar control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a progress bar control.

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes

# **CPropertyPage Class**

# Description

This class is used to create and manage a property page A property sheet will have one or more property pages. These pages are much like dialogs which are presented within a tabbed dialog or within a wizard. The data on a property page can be validated before the next page is presented.

Refer to the PropertySheet demo program for an example of how property pages can be used.

# **CPropertyPage Members**

#### Construction

CPropertyPage					
	<b>CPropertyPage</b>	(UINT	nIDTemplate,	LPCTSTR	szī
	. , , ,				
	The constructor for CPropertyPage.				

#### **Attributes and State functions**

GetPSP	<pre>PROPSHEETPAGE GetPSP() const {return m_PSP;}</pre>
	Retrieves the PROPSHEETPAGE struct for this property page.
	$POOL T_{c}P_{u}$ + to $P_{c}$ + to $P_{u}$
IsButtonEnabled	BOOL IsButtonEnabled(int iButton) const;
	Returns TRUE if the specified button is active.
SetModified	
	<pre>void SetModified(BOOL bChanged) const;</pre>
	Flags a property page as modified or unmodified.

#### **Overridables**

D	ialogProc	virtual BOOL DialogProc(UINT uMsg, WPARAM wParam, LPARAM l
		Override this function to handle messages received by the property page.

OnApply	<pre>virtual int OnApply();</pre>
	Called when the Apply button is pressed.
OnCancel	<pre>virtual void OnCancel();</pre>
	Called when the Cancel button is pressed.
OnHelp	<pre>virtual void OnHelp();</pre>
1	Called when the Help button is pressed.
OnInitDialog	<pre>virtual BOOL OnInitDialog();</pre>
0	Called when the property page is created.
OnKillActive	<pre>virtual BOOL OnKillActive();</pre>
	Called when the property page becomes inactive in response to an OK or Apply but
OnNotify	<pre>virtual LRESULT OnNotify(WPARAM wParam, LPARAM lParam);</pre>
5	Called when a notification from a child control is received.
OnOK	<pre>virtual int OnOK();</pre>
	Called when the OK button is pressed.
OnQueryCancel	<pre>virtual BOOL OnQueryCancel();</pre>
	Called when the Cancel button is press, before the cancel has taken place.
	virtual BOOL OnQuerySiblings(WPARAM wParam, LPARAM lPara
OnQuerySiblings	Called in response to a query from the property sheet. Return zero to indicate passed indicated failed.
OnSetActive	<pre>virtual int OnSetActive();</pre>
	Called when the property page becomes active.
OnWizardBack	<pre>virtual int OnWizardBack();</pre>
	Called when the Back button is pressed on a wizard.
OnWizardFinish	<pre>virtual BOOL OnWizardFinish();</pre>
	Called when the Finish button is pressed on a wizard.
OnWizardNext	<pre>virtual int OnWizardNext();</pre>
	Called when the Next button is pressed on a wizard.

#### virtual BOOL PreTranslateMessage(MSG\* pMsg);

PretranslateMessage

Called allow the tab control to translate keyboard input before the message is passed on message loop.

## Operations

CancelToClose	<pre>void CancelToClose() const;</pre>
	Disables the Cancel button and changes the text of the OK button to "Close."
QuerySiblings	LRESULT QuerySiblings(WPARAM wParam, LPARAM lParam) const;
	Used to query other property pages belonging to this property sheet.
SetTitle	<pre>void SetTitle(LPCTSTR szTitle);</pre>
	Sets the property sheet's title.
SetWizardButtons	<pre>void SetWizardButtons(DWORD dwFlags) const;</pre>
	Sets the wizard buttons.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

Header file	propertysheet.h
Win32/64 support	Yes
WinCE support	Yes

# **CPropertySheet Class**

# Description

This class is used to create and manage a property sheet. A property sheet will have one or more property pages. These pages are much like dialogs which are presented within a tabbed dialog or within a wizard. The data on a property page can be validated before the next page is presented. Property sheets have three modes of use: Modal, Modeless, and Wizard.

Refer to the PropertySheet demo program for an example of how property sheets can be used.

# **CPropertySheet Members:**

#### Construction

CPropertySheet	CPropertySheet(UINT nIDCaption, HWND hwndPa
	<sup>t</sup> CPropertySheet(LPCTSTR pszCaption = NULL, H
	Constructs a CPropertySheet object.

#### Attributes

GetActivePage	CPropertyPage* GetActivePage() const; Retrieves the active CPropertyPage.
GetPageCount	<pre>int GetPageCount() const; Retrieves the number of pages in the property sheet.</pre>
GetPageIndex	<pre>int GetPageIndex(CPropertyPage* pPage) const; Retrieves the index of the specified page</pre>
GetTabControl	HWND GetTabControl() const; Retrieves the window handle of the property sheet's tab control.
	<pre>virtual BOOL SetActivePage(int nPage);</pre>

SetActivePage	<pre>virtual BOOL SetActivePage(CPropertyPage* pPage);</pre>
	Sets the active property page.
SetWizardMode	virtual vaid CathlizardMada(ROOL bhlizard).
	<pre>virtual void SetWizardMode(BOOL bWizard);</pre>
	Set to TRUE if the property sheet is a wizard.

## **State functions**

IsModeless	BOOL IsModeless() const;
	Returns TRUE if the property sheet is modeless.
IsWizard	BOOL IsWizard() const;
	Returns TRUE if the property sheet is a wizard.

### **Overridables**

Doctroy	<pre>virtual void Destroy();</pre>
Destroy	
	Called when CPropertySheet is about to be deconstructed.
	<pre>virtual BOOL PreTranslateMessage(MSG* pMsg);</pre>
PreTranslateMessag	
	Translates keyboard and mouse messages before they are passed to
	TranslateMessage.

# Operations

AddPage	<pre>virtual CPropertyPage* AddPage(CPropertyPage* pPage);</pre>
Auur age	Adds a property page to the property sheet.
Create	<pre>virtual HWND Create(HWND hWndParent = 0);</pre>
Cledle	Creates a modeless property sheet.
CreatePropertySheet	<pre>virtual INT_PTR CreatePropertySheet(LPCPROPSHEETHEADER pps</pre>
Creater roperty snee	Creates a property sheet.
DestroyButton	<pre>virtual void DestroyButton(int iButton);</pre>
Desitoy Dutton	

	Removes a button.
DoModal	<pre>virtual int DoModal(); Creates a modal property sheet.</pre>
RemovePage	<pre>virtual void RemovePage(CPropertyPage* pPage); Remove a property page.</pre>
SetTitle	<pre>virtual void SetTitle(LPCTSTR szTitle); Sets the title of the property sheet.</pre>

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

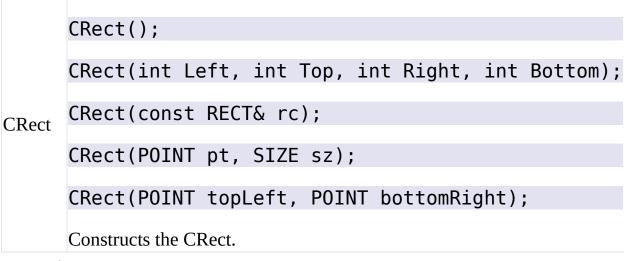
Header file	propertysheet.h
Win32/64	Yes
support	
WinCE support	Yes

# **CRect Class**

## Description

A CRect can be used anywhere that a RECT structure could be used.

# **CRect Members**



#### operator =

```
void operator=(const RECT& srcRect);
```

```
Assigns a RECT to this CRect. Operator ()
```

```
operator LPRECT();
```

Returns a pointer to the RECT associated with this CRect. **Operator** ()

operator LPCRECT() const;

Returns a const pointer to the RECT associated with this CRect. Operator ==

```
BOOL operator == (const RECT& rc) const;
```

Returns TRUE if the co-ordinates of the source rectangle and the CRect are equal. **Operator** !=

BOOL operator != (const RECT& rc) const;

Returns TRUE if the of the source rectangle and the CRect are not equal. Operator +=

void operator += (POINT pt);

void operator += (SIZE size);

void operator += (LPCRECT prc);

Adds the specified values to the CRect. **Operator** -=

void operator -= (POINT pt);

void operator -= (SIZE sz);

void operator -= (LPCRECT prc);

Subtracts the specified values from the CRect. Operator &=

void operator &= (const RECT& rc);

Determines the intersection with the specified RECT. Operator  $\mid$  =

void operator |= (const RECT& rc);

Determines the union with the specified RECT. **OPERATOR** +

CRect operator + (POINT pt) const;

CRect operator + (SIZE sz) const;

CRect operator + (LPRECT prc) const;

Offsets the CRect and returns the result. **Operator** -

CRect operator - (POINT pt) const;

CRect operator - (SIZE sz) const;

CRect operator - (LPRECT prc) const;

Offsets the CRect and returns the result. Operator &

CRect operator & (const RECT& rc) const;

Returns the intersection with the specified RECT. **Operator** 

CRect operator | (const RECT& rc) const;

Returns the union with the specified RECT. CopyRect

BOOL CopyRect(LPCRECT prc);

Copies the coordinates of the source rectangle to the CRect. DeflateRect

B00L DeflateRect(int x, int y);

B00L DeflateRect(SIZE size);

void DeflateRect(LPCRECT prc);

void DeflateRect(int l, int t, int r, int b);

Decreases the width and height of the CRect. EqualRect

B00L EqualRect(LPRECT prc);

Determines whether the source rectangle and the CRect are equal by comparing the coordinates of their upper-left and lower-right corners. InflateRect

B00L InflateRect(int dx, int dy);

B00L InflateRect(SIZE sz);

void InflateRect(LPCRECT prc);

void InflateRect(int l, int t, int r, int b);

Increases the width and height of the CRect. IntersectRect

B00L IntersectRect(LPCRECT prc1, LPCRECT prc2);

Calculates the intersection of two source rectangles and places the coordinates of the intersection rectange into the CRect. IsRectEmpty

B00L IsRectEmpty() const;

Determines whether the CRect is empty. IsRectNull

BOOL IsRectNull() const;

Determines whether the CRect is NULL. MoveToX

void MoveToX (int x);

Moves the rect to the specified left position. MoveToY

void MoveToY (int y);

Moves the rect to the specified top position. MoveToXY

void MoveToXY (int x, int y);

void MoveToXY (POINT pt);

Moves to rect to the specified left and top positions. MulDiv

CRect MulDiv(int nMult, int nDiv) const;

Multiplies the CRect values by nMult, and then divides the result by nDiv. NormalizeRect

void NormalizeRect()

Normalizes CRect so that both the height and width are positive. OffsetRect

B00L OffsetRect(int dx, int dy);

B00L OffsetRect(POINT pt);

B00L OffsetRect(SIZE size);

Moves the CRect by the specified offsets. PtInRect

BOOL PtInRect(POINT pt) const;

Determines whether the specified point lies within the CRect. SetRect

BOOL SetRect(int left, int top, int right, int bottom);

BOOL SetRect(POINT TopLeft, POINT BtmRight);

Sets the coordinates of the CRect. SetRectEmpty

BOOL SetRectEmpty();

Sets all the coordinates of the CRect to zero. SubtractRect

B00L SubtractRect(LPCRECT prc1, LPCRECT prc2);

Sets the coordinates of the CRect to those formed by subtracting one rectangle from another. UnionRect

B00L UnionRect(LPCRECT prc1, LPCRECT prc2);

Creates the union of two rectangles. Height

int Height() const;

Returns the height of the CRect. Width

int Width() const;

Returns the width of the CRect. Size

CSize Size() const;

Determines the size (width and height) of the CRect. CenterPoint

```
CPoint CenterPoint() const;
```

Returns the point at the center of the CRect. TopLeft

CPoint TopLeft() const;

Returns the top left point of the CRect. BottomRight

```
CPoint BottomRight() const;
```

Returns the bottom left point of the CRect.

### Remarks

By default, the constructor sets all the rectangle coordinates zero, but the CRect can also be constructed from four integers, a rectangle, a point and size, or two points.

The CRect class inherits from the RECT structure. As a result, the data members of the underlying RECT struct are also accessible as data members (left, top, right and bottom).

Header file	winutils.h
Win32/64 support	Yes
WinCE support	Yes

# **CResizer Class**

# Description

The CResizer class can be used to automatically rearrange the position of child windows when the parent window is resized. Typically this is used to reposition the controls for a dialog.

# **CResizer Members**

CResizer

CResizer();

Constructor for the CResizer.

# AddChild

virtual void AddChild(CWnd\* pWnd, Alignment corner, DWORD dwStyle);

virtual void AddChild(HWND hWnd, Alignment corner, DWORD dwStyle);

Adds a child window to be managed by the CResizer. GetMinRect

```
CRect GetMinRect() const;
```

Returns the minimum allowed rectangle. GetMaxRect

CRect GetMaxRect() const;

Returns the maximum allowed rectangle. HandleMessage

virtual void HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam);

Performs the resizing and scrolling. Call this function from within the window's DialogProc. **Initialize** 

virtual void Initialize(CWnd\* pParent, RECT rcMin, RECT rcMax = CRect(0,

Specifies the parent window, as well as its minimum and maximum size. RecalcLayout

virtual void RecalcLayout();

Repositions the child windows.

# Remarks

To use the CResizer class to manage the position of child windows, perform the following steps.

- Use Initialize to specify the parent window, along with the minimum and maximum permitted sizes.
- Use AddChild for each child window we wish to manage.
- Call HandleMessage from within DialogProc or WndProc to pass messages on to CResizer

The following code demonstrates how to initialize CResizer for a dialog.

```
BOOL CMyDialog::OnInitDialog()
{
    // Perform other tasks ...
    // Initialize dialog resizing
    m Resizer.Initialize( this, CRect(0, 0, 300, 200) );
    m Resizer.AddChild(m RadioA,
                                   topleft, 0);
    m Resizer.AddChild(m RadioB,
                                   topleft, 0);
    m Resizer.AddChild(m RadioC,
                                   topleft, 0);
    m Resizer.AddChild(m Button,
                                   topleft, 0);
    m Resizer.AddChild(m CheckA,
                                   bottomright, 0);
    m Resizer.AddChild(m CheckB,
                                   bottomright, 0);
    m Resizer.AddChild(m CheckC,
                                   bottomright, 0);
    m_Resizer.AddChild(m_RichEdit1, topright, RD_STRETCH_WIDTH);
    m Resizer.AddChild(m RichEdit2, bottomleft, RD STRETCH WIDTH| RD STR
    return true;
```

The following code shows how to add the CResizer message handling.

```
INT_PTR CMyDialog::DialogProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    // Pass resizing messages on to the resizer
    m_Resizer.HandleMessage(uMsg, wParam, lParam);
// switch (uMsg)
// {
    // Additional messages to be handled go here
// }
    // Pass unhandled messages on to parent DialogProc
    return DialogProcDefault(uMsg, wParam, lParam);
```

Header file	dialog.h
Win32/64 support	Yes
WinCE support	Yes

# **CRgn** Class

## Description

The class responsible for creating and managing regions.

# **CRgn Members**

Initializati	on and Assignment
CRgn	CRgn(); CRgn(HRGN_hRgn); Constructs a CRgn object.
<b>D</b> ecomo I I a	

## FromHandle

```
static CRgn* FromHandle(HRGN hRgn);
```

Returns the CRgn associated with the specified region handle. If a CRgn object doesn't already exist, a temporary CRgn object is created. This temporary CRgn will be deleted sometime after the processing of

the current message is complete. **Operator HRGN()** 

#### operator HRGN() const;

Allows a CRgn object to be used as a region handle (HRGN). Attributes GetRegionData

int GetRegionData(LPRGNDATA lpRgnData, int nDataSize) const;

Fills the specified buffer with data describing a region. GetRgnBox

int GetRgnBox(RECT& rc) const;

Retrieves the bounding rectangle of the region, and stores it in the specified RECT. The return value indicates the region's complexity: NULLREGION;SIMPLEREGION; or COMPLEXREGION. **Operations** 

### CombineRgn

int CombineRgn(HRGN hRgnSrc1, HRGN hRgnSrc2, int nCombineMode);

int CombineRgn(HRGN hRgnSrc, int nCombineMode);

Combines two sepcified regions and stores the result. CopyRgn

int CopyRgn(HRGN hRgnSrc);

Assigns the specified region to the current region. CreateEllipticRgn

HRGN CreateEllipticRgn(int x1, int y1, int x2, int y2);

Creates an elliptical region. CreateEllipticRgnIndirect

HRGN CreateEllipticRgnIndirect(const RECT& rc);

 $\label{eq:creates} {\tt Creates an elliptical region.} \ CreateFromData$ 

Creates a region from the specified region and transformation data. CreateFromPath

HRGN CreateFromPath(HDC hDC);

Creates a region from the path that is selected into the specified device context. The resulting region uses device coordinates. CreatePolygonRgn

HRGN CreatePolygonRgn(LPPOINT lpPoints, int nCount, int nMode);

Creates a polygonal region. CreatePolyPolygonRgn

Creates a region consisting of a series of polygons. EqualRgn

BOOL EqualRgn(HRGN hRgn) const;

Creates a region consisting of a series of polygons. The polygons can overlap. OffsetRgn

int OffsetRgn(int x, int y);

int OffsetRgn(POINT& pt);

Moves a region by the specified offsets. PtInRegion

B00L PtInRegion(int x, int y) const;

B00L PtInRegion(P0INT& pt) const;

Determines whether the specified point is inside the specified region. **RectInRegion** 

B00L RectInRegion(const RECT& rc) const;

Determines whether the specified rect is inside the specified region. SetRectRgn

void SetRectRgn(int x1, int y1, int x2, int y2);

void SetRectRgn(const RECT& rc);

Converts the region into a rectangular region with the specified coordinates.

**Base class Members** 

For base class members, refer to the members of <u>CGDIObject</u>.

### Remarks

CRgn objects can be used anywhere a a handle to a region (HRGN) might be used. They can be substituted for the HRGN in any of the Windows API functions which use a HRGN as a function argument. The benefit of using a CRgn object is that it automatically deletes the palette when it is destroyed.

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64 support	Yes
WinCE support	Yes

# **CRibbon Class**

# Description

CRibbon allows a ribbon to be added to a windows. Only Windows 7 (and above) supports the ribbon framework.

# **CRibbon Members**

CRibbon();

Constructs a CRibbon object.

CreateRibbon

bool virtual CreateRibbon(CWnd\* pWnd);

Creates the ribbon DestroyRibbon

void virtual DestroyRibbon();

Destroys the ribbon GetRibbonFramework

IUIFramework\* GetRibbonFramework() const;

Returns the pointer to IUIFramwork, assigned when the ribbon is created. GetRibbonHeight

UINT GetRibbonHeight() const;

Returns the height of the ribbon.

### **IUnknown Methods**

AddRef	<pre>STDMETHOD_(ULONG, AddRef());</pre>
	Increments the reference count for the interface on the object.
Release	<pre>STDMETHOD_(ULONG, Release());</pre>
	Decrements the reference count for the interface on the object.

QueryInterface STDMETHOD(QueryInterface(REFIID iid, void\*\* ppv));

Retrieves pointers to the supported interfaces on the object.

### **IUIApplication Methods**

OnCreateUICommand	STDMETHOD(OnCreateUICommand)(UINT nCmdID,in UI_COMMANderef_out IUICommandHandler** ppCommandHar Called by the Ribbon framework for each command specified in the Ribbon marku
	IUICommandHandler.
OnDestroyUICommand	STDMETHOD(OnDestroyUICommand)(UINT32 commandId,in UIin_opt IUICommandHandler* commandHandler); Called for each Command specified in the Ribbon markup when the Ribbon host ar destroyed.
OnViewChanged	STDMETHOD(OnViewChanged)(UINT viewId,in UI_VIEWTYPE 1 in IUnknown* pView, UI_VIEWVERB verb, INT Called when the state of the Ribbon changes, for example, created, destroyed, or re

#### **IUICommandHandle methods**

	<pre>STDMETHODIMP Execute(UINT nCmdID, UI_EXECUTIONVERB verb,in</pre>	
Execute	in_opt const PROPVARIANT* ppropvarValu	
Execute	in_opt IUISimplePropertySet* pCommandE	
	Executes or previews the Commands bound to the Command handler.	
UpdateProperty	STDMETHODIMP UpdateProperty(UINT nCmdID,in REFPROPERTYKEY	
	in_opt const PROPVARIANT* pprop	
	out PROPVARIANT* ppropvarNewVal	
	Called by the Ribbon framework when a command property (PKEY) needs to be updated.	

## Remarks

Microsoft's RibbonUI framework uses COM to implement the ribbon. The COM interfaces involved are IUIApplication and IUICommandHandler. The CRibbon class inherits from both IUIApplication and IUICommandHandler.

To create and interact with the ribbon, we override the relevant functions from both IUIApplication and IUICommandHandler.

The functions you may wish to override are:

- IUIApplication::OnCreateUICommand Called for each Command specified in the Ribbon markup to bind the Command to an IUICommandHandler.
- IUIApplication::OnDestroyUICommand Called for each Command specified in the Ribbon markup when the Ribbon host application window is destroyed.
- IUIApplication::OnViewChanged Called when the state of a View changes.
- IUICommandHandler::Execute Executes or previews the Commands bound to the Command handler.
- IUICommandHandler::UpdateProperty Sets a property value for a bound Command, for example, setting a Command to enabled or disabled depending on the state of a View.

Refer to the RibbonSimple sample for a demonstration of how to add a ribbon to a simple window.

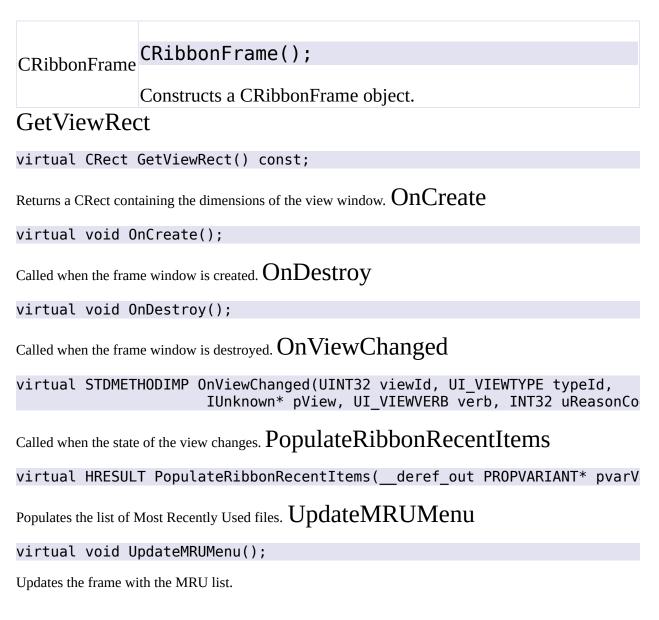
Header file	ribbon.h
Win32/64 support	Yes
WinCE support	No
Library required	Comctl32.lib, Shlwapi.lib

# **CRibbonFrame Class**

## Description

CRibbonFrame provides a Frame window with a Ribbon in place of the traditional MenuBar and ToolBar. Only Windows 7 (and above) supports the ribbon framework. By default, a ToolBar and MenuBar is displayed instead of the ribbon if the ribbon framework is not supported by the Operating System.

## **CRibbonFrame Members**



#### **Base class Members**

For base class members, refer to the members of CFrame and CRibbon.

# Remarks

Microsoft's RibbonUI framework uses COM to implement the ribbon. The COM interfaces involved are IUIApplication and IUICommandHandler. The Win32++ CRibbon class inherits from both IUIApplication and IUICommandHandler. CRibbonFrame inherits from both CFrame and and CRibbon.

To add a ribbon to your frame, inherit CMainFrame from CRibbonFrame, and add the Ribbon.xml file containing the definitions for your ribbon to the project. The Ribbon.xml is compiled to produce the RibbonUI.h and Ribbon.rc before the files C++ files are compiled. To compile Ribbon.xml, specify it's custom build properties in the IDE as follows:

- Command Line:- uicc.exe ..\src\Ribbon.xml ..\src\Ribbon.bml /header:..\src\RibbonUI.h /res:..\src\RibbonUI.rc
- Outputs:- Ribbon.bml;RibbonUI.rc;RibbonUI.h

To create and interact with the ribbon, we override the relevant functions from both IUIApplication and IUICommandHandler. The functions you may wish to override are:

- IUIApplication::OnCreateUICommand Called for each Command specified in the Ribbon markup to bind the Command to an IUICommandHandler.
- IUIApplication::OnDestroyUICommand Called for each Command specified in the Ribbon markup when the Ribbon host application window is destroyed.
- IUIApplication::OnViewChanged Called when the state of a View changes.
- IUICommandHandler::Execute Executes or previews the Commands bound to the Command handler.
- IUICommandHandler::UpdateProperty Sets a property value for a bound Command, for example, setting a Command to enabled or disabled depending on the state of a View.

Refer to the RibbonFrame sample for a demonstration of how to create an application with a ribbon frame.

Header file	ribbon.h
Win32/64 support	Yes
WinCE support	No
Library required	Comctl32.lib, Shlwapi.lib

# **CRichEdit Class**

## Description

A rich edit control enables the user to enter, edit, print, and save text. The text can be assigned character and paragraph formatting, and can include embedded Component Object Model (COM) objects.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of rich edit controls.

## **CRichEdit Members**

#### Construction

CRichEdit	CRichEdit();
	Constructor for CRichEdit.

### Attributes

GetCharPos	CPoir	nt GetCharPos(long lChar) const;	
	Retrieves the client area coordinates of a specified character.		
		<pre>DWORD GetDefaultCharFormat(CHARFORMAT&amp; cf) const;</pre>	
GetDefaultChar	Format	<pre>GetDefaultCharFormat(CHARFORMAT2&amp; cf) const;</pre>	
		Retrieves the current default character formatting attributes.	
	GetE	ventMask() const;	
GetEventMask	Retrieves the event mask. The event mask specifies which notification messages the control sends to its parent window.		
GetFirstVisibleLine() const;		t GetFirstVisibleLine() const;	
	Ge	ts the zero-based index of the uppermost visible line.	
GetIRichEditOl		<pre>chEdit0le* GetIRichEdit0le() const;</pre>	

		Retrieves an IRichEditOle object that a client can use to access the rich edit control's Component Object Model (COM) functionality.	
GetFirstV	Visible	int GetFirstVisibleLine() const; Line Returns the zero-based index of the first visible character in a single-line edit control or the zero-based index of the uppermost visible line in a multiline edit control.	
GetLimit	Text	long GetLimitText() const; Gets the current text limit for the edit control.	
	int	<pre>GetLine(int nIndex, LPTSTR lpszBuffer) const;</pre>	
CotI ino	int	GetLine(int nIndex, LPTSTR lpszBuffer, int nMaxLength) cons	
Сс	pointe	es a line of text from the rich edit control and places it in the specified buffer. lpszBuffer is er to the buffer that receives a copy of the line. Either specify nMaxLength, or set the first of lpszBuffer to the size of the buffer in TCHARs.	
GetLine	ount	<pre>int GetLineCount() const;</pre>	
GetLine	Jouint	Gets the number of lines in a multiline edit control.	
GetModify BOOL GetModify() const; Retrieves a flag than indicates whether the contents of the edit control have modified		Retrieves a flag than indicates whether the contents of the edit control have been	
GetOptio	ms	UINT GetOptions() const;	
Getopho	/115	Retrieves the rich edit control options.	
		<pre>DWORD GetParaFormat(PARAFORMAT&amp; pf) const;</pre>	
GetParaF	Format	DWORD GetParaFormat(PARAFORMAT2& pf) const;	
		Retrieves the paragraph formatting of the current selection.	
		<pre>void GetRect(LPRECT lpRect) const;</pre>	
GetRect		Retrieves the formatting rectangle. The formatting rectangle is the limiting rectangle into which text can be drawn.	
GetRedoName	Name	UNDONAMEID GetRedoName() const;	
GetKeuoIname		Retrieves the type of the next action, if any, in the control's redo queue.	
		<pre>void GetSel(CHARRANGE&amp; cr) const;</pre>	
GetSel void GetSel(long& nStartChar, long& nEndChar) const;			

	Retrieves the starting and ending character positions of the selection	
DWORD GetSelectionCharFormat(CHARFORMAT& cf) const; GetSelectionCharFormat DWORD GetSelectionCharFormat(CHARFORMAT2& cf) const;		
	Retrieves the character formatting attributes in the current selection.	
GetSelectionTy		
	Retrieves the type of contents in the current selection.	
	<pre>long GetSelText(LPSTR lpBuf) const;</pre>	
GetSelText	CString GetSelText() const;	
	Gets the text of the current selection.	
	<pre>long GetTextLength() const;</pre>	
GetTextLength	Retrieves the length of the text, in characters. Does not include the terminating null character.	
GetTextLengthl		
	Returns the number of TCHARs in the rich edit control, depending on the flags specified.	
GetTextMode	UINT GetTextMode() const;	
	Retrieves the current text mode and undo level.	
GetTextRange	int GetTextRange(int nFirst, int nLast, CString& refString) co	
I	Retrieves the specified range of text.	
GetUndoName	UNDONAMEID GetUndoName() const;	
Getonuorvanie	Returns.	
SetAutoURLDe	BOOL SetAutoURLDetect(BOOL bEnable = TRUE) const;	
Indicates if the auto URL detection is active.		
SetBackground	Color COLORREF SetBackgroundColor(BOOL bSysColor, COLORREF cr) (	
-		
	Sets the background color.	
	B00L SetDefaultCharFormat(CHARF0RMAT& cf) const;	
SetDefaultChar	B00L SetDefaultCharFormat(CHARF0RMAT& cf) const;	

SetEventMask	DWORD SetEventMask(DWORD dwEventMask) const;		
	Sets the event mask. The event mask specifies which notification messages the control sends to its parent window.		
	<pre>void SetModify(BOOL bModified = TRUE) const;</pre>		
SetModify	Sets or clears the modification flag. The modification flag indicates whether the text has been modified.		
SetOLECallbac	k BOOL SetOLECallback(IRichEditOleCallback* pCallback) const; Sets the IRichEditOleCallback COM object.		
SetOptions	<pre>void SetOptions(WORD wOp, DWORD dwFlags) const; Sets the options. Possible wOp values: ECOOP_SET, ECOOP_OR, ECOOP_AND, ECOOP_XOR Possible dwFlags: ECO_AUTOWORDSELECTION, ECO_AUTOVSCROLL, ECO_AUTOHSCROLL, ECO_NOHIDESEL ECO_READONLY, ECO_WANTRETURN, ECO_SELECTIONBAR, ECO_VERTICAL</pre>		
	BOOL SetParaFormat(PARAFORMAT& pf) const;		
SetParaFormat	BOOL SetParaFormat(PARAFORMAT2& pf) const;		
	Sets the paragraph formatting attributes in the current selection.		
SetReadOnly	B00L SetReadOnly(B00L bReadOnly = TRUE) const;		
5	Sets or removes the read-only style.		
	<pre>void SetRect(LPCRECT lpRect) const;</pre>		
SetRect	Sets the formatting rectangle. The formatting rectangle is the limiting rectangle into which the control draws the text.		
	<pre>void SetSel(long nStartChar, long nEndChar) const;</pre>		
SetSel	<pre>void SetSel(CHARRANGE&amp; cr) const;</pre>		
Selects a range of characters.			
	B00L SetSelectionCharFormat(CHARFORMAT& cf) const;		
SetSelectionCharFormat B00L SetSelectionCharFormat(CHARFORMAT2& cf) const;			
	Sets the character formatting attributes in the current selection.		
	BOOL SetTargetDevice(HDC hDC, long lLineWidth) const;		
SetTargetDevic	<pre>eBOOL SetTargetDevice(CDC&amp; dc, long lLineWidth) const;</pre>		

	Sets the target output device and line width used for "what you see is what you get" (WYSIWYG) formatting.		
SetTextMode	BOOL SetTextMode(UINT fMode) const;		
	Sets the text mode or undo level of the rich edit control. The message fails if the control contains text.		
	UINT SetUndoLimit(UINT nLimit) const;		
SetUndoLimit	Sets the maximum number of actions that can stored in the undo queue. This member function fails if the control contains text.		
	BOOL SetWordCharFormat(CHARFORMAT& cf) const;		
SetWordCharFo			
	<pre>ormat BOOL SetWordCharFormat(CHARFORMAT2&amp; cf) const;</pre>		
	Sets the character formatting attributes in the current word.		

# Operations

CanPaste	BOOL CanPaste(UINT nFormat = 0) const; Determines whether a rich edit control can paste a specified clipboard format.		
CanRedo BOOL CanRedo() const;			
CanUndo	Determines whether there are any actions in the control redo queue. BOOL CanUndo() const; Determines whether there are any actions in an edit control's undo queue.		
CharFromPos	<pre>int CharFromPos(CPoint pt) const; Gets information about the character closest to a specified point in the client area of an equilation</pre>		
Clear	<pre>void Clear() const; Delete (clear) the current selection</pre>		
Сору	<pre>void Copy() const; Copy the current selection to the clipboard in CF_TEXT format.</pre>		
Cut	<pre>void Cut() const; Delete (cut) the current selection, if any, in the edit control and copy the deleted text to th format.</pre>		

DisplayBand BOOL DisplayBand(LPRECT pDisplayRect) const;	
	Displays a portion of the contents of a rich edit control, as previously formatted for EM_FORMATRANGE message.
	<pre>void EmptyUndoBuffer() const;</pre>
EmptyUndoBuffer	Resets the undo flag of the rich edit control. The undo flag is set whenever an opera undone.
FindText	<pre>long FindText(DWORD dwFlags, FINDTEXTEX* pFindTextEx) co</pre>
	Finds text within the rich edit control.
	DWORD FindWordBreak(UINT nCode, DWORD nStart) const;
FindWordBreak	Finds the next word break before or after the specified character position or retrieve that position.
FormatRange	<pre>long FormatRange(FORMATRANGE* pfr, BOOL bDisplay = TRUE)</pre>
C	Formats a range of text in a rich edit control for a specific device (e.g. printer).
HideSelection	<pre>void HideSelection(BOOL bHide, BOOL bPerm) const;</pre>
	Shows or hides the current selection.
LimitText	<pre>void LimitText(int nChars = 0) const;</pre>
	Limits the amount of text a user can enter.
LineFromChar	<pre>long LineFromChar(long nIndex) const;</pre>
	Determines which line contains the given character.
LineIndex	<pre>int LineIndex(int nLine = -1) const;</pre>
Linemacx	Retrieves the character index of a given line.
LineLength	<pre>int LineLength(int nLine = -1) const;</pre>
LIIIeLeiigui	Retrieves the length of a given line.
LineScroll	<pre>void LineScroll(int nLines, int nChars = 0) const;</pre>
Linescion	Scrolls the text.
Paste	<pre>void Paste() const;</pre>
i uote	Inserts the contents of the Clipboard.
PasteSpecial	<pre>void PasteSpecial(UINT nClipFormat, DWORD dwAspect = 0,</pre>

	Inserts the contents of the Clipboard in the specified data format.
PosFromChar	CPoint PosFromChar(UINT nChar) const;
	Retrieves the client area coordinates of a specified character.
Redo	BOOL Redo() const;
	Redoes the next action in the control's redo queue.
ReplaceSel	<pre>void ReplaceSel(LPCTSTR lpszNewText, B00L bCanUndo) const;</pre>
Replaceder	Replaces the current selection with specified text.
RequestResize	<pre>void RequestResize() const;</pre>
requestresize	Forces the sending of a request resize notifications.
	<pre>void StopGroupTyping() const;</pre>
StopGroupTyping	Stops the control from collecting additional typing actions into the current undo action. T action, if any, into a new action in the undo queue.
StreamIn	<pre>long StreamIn(int nFormat, EDITSTREAM&amp; es) const;</pre>
Streamin	Replaces text with text from the specified input stream.
StreamOut	<pre>long StreamOut(int nFormat, EDITSTREAM&amp; es) const;</pre>
	Stores text into an output stream.
Undo	<pre>void Undo() const;</pre>
Chuo	Reverses the last editing operation.

## Overidables

	<pre>virtual void PreRegisterClass(WNDCLASS &amp;wc);</pre>
	Set the window class parameters before the window is created.

### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Like all common controls, the rich edit control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for an edit control.

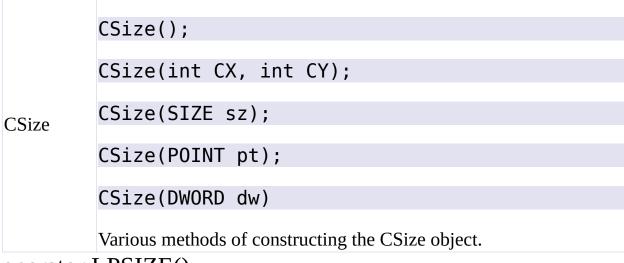
Header file	rich_edit.h
Win32/64 support	Yes
WinCE support	NO
Library required	Comctl32.lib

# **CSize Class**

## Description

A CSize can be used anywhere that a SIZE structure could be used.

## **CSize Members**



## operator LPSIZE()

operator LPSIZE();

Returns the pointer to the SIZE object associated with this CSize. **Operator** ==

BOOL operator == (SIZE sz) const;

Returns TRUE if the co-ordinates of the source size and the CSize are equal. Operator !=

BOOL operator != (SIZE sz) const;

Returns TRUE if the of the source size and the CSize are not equal. Operator +=

void operator += (SIZE sz);

Adds the specified SIZE. **Operator** -=

void operator -= (SIZE sz);

Subtracts the specified SIZE. **Operator** -

CSize operator -	()	const;
------------------	----	--------

Returns the unary minus (additive inverse). **Operator** +

CSize operator + (SIZE sz) const;

CPoint operator + (POINT point) const;

CRect operator + (LPCRECT prc) const;

Adds the point and returns the value. **Operator** -

CSize operator - (SIZE sz) const;

CPoint operator - (POINT point) const;

CRect operator - (LPCRECT prc) const;

Subtracts the point and returns the value. SetSize

void SetSize(int CX, int CY);

Sets the coordinates of the CSize.

### Remarks

By default, the constructor sets all the size coordinates zero, but the CSize can also be constructed from two integers, a size, a point, or a dword.

The CSize class inherits from the SIZE structure. As a result, the data members of the underlying SIZE struct are also accessible as data members.

Header file	winutils.h
Win32/64 support	Yes
WinCE support	Yes

# **CSlider** Class

## Description

The CSlider class is used to create and manage a slider control.

## **CSlider Members**

CSlider();

Constructor for the CSlider.

ClearSel

void ClearSel() const;

Clears the current selection range ClearTics

void ClearTics(BOOL bRedraw = FALSE ) const;

Removes the current tick marks GetBuddy

HWND GetBuddy(BOOL fLocation = TRUE ) const;

Retrieves the handle to the trackbar control buddy window at the given location.

## GetChannelRect

CRect GetChannelRect() const;

Retrieves the size and position of the bounding rectangle fro the trackbar's channel. GetLineSize

int GetLineSize() const;

Retrieves the number of logical positions the trackbar's slide moves in response to keyboard input from the arrow keys. GetNumTics

int GetNumTics() const;

Retrieves the number of tick marks in the trackbar. GetPageSize

#### int GetPageSize() const;

Retrieve the number of logical positions the trackbar's slider move in response to keyboard input, or mouse input such as clicks in the trackbar's channel. GetPos

#### int GetPos() const;

Retrieves the current logical position of the slider in the trackbar. GetRangeMax

int GetRangeMax() const;

Retrieves the maximum position for the slider in the trackbar. GetRangeMin

int GetRangeMin() const;

Retrieves the minimum position for the slider in the trackbar. GetSelEnd

int GetSelEnd() const;

Retrieves the ending position of the current selection range in the trackbar. GetSetStart

int GetSelStart() const;

Retrieves the starting position of the current selection range in the trackbar. GetThumbLength

int GetThumbLength() const;

Retrieves the length of the slider in the trackbar. GetThumbRect

CRect GetThumbRect() const;

Retrieve the size and position of the bounding rectangle for the slider in the trackbar. GetTic

int GetTic(int nTic ) const;

Retrieves the logical position of a tick mark in the trackbar. GetTicPos

int GetTicPos(int nTic) const;

Retrieves the current physical position of a tick mark in the trackbar. GetToolTips

HWND GetToolTips() const;

Retrieves the handle to the ToolTip control assigned to the trackbar, if any. SetBuddy

HWND SetBuddy(HWND hwndBuddy, BOOL fLocation = TRUE ) const;

Assigns a window as the buddy window for the trackbar control. SetLineSize

int SetLineSize(int nSize) const;

Sets the number of logical positions the trackbar's slider moves in response to keyboard input from arrow keys. SetPageSize

int SetPageSize(int nSize) const;

Sets the number of logical positions the trackbar's slider moves in response to keyboard input, or mouse input such as clicks in the trackbar's channel. SetPos

void SetPos(int nPos, BOOL bRedraw = FALSE) const;

Sets the current logical position of the slider in the trackbar. SetRangeMax

void SetRangeMax(int nMax, BOOL bRedraw = FALSE) const;

Set the maximum logical position for the slider in the trackbar. SetRangeMin

void SetRangeMin(int nMax, BOOL bRedraw = FALSE) const;

Sets the minimum logical position for the slider in the trackbar. SetSelection

void SetSelection(int nMin, int nMax, BOOL bRedraw = FALSE) const;

Sets the starting and ending positions for the available selection range in the trackbar. SetTic

B00L SetTic(int nTic) const;

Sets a tick mark in the trackbar at the specified logical position. SetTicFreq

void SetTicFreq(int nFreq) const;

Sets the interval frequency for tick marks in the trackbar. SetTipSide

int SetTipSide(int nLocation) const;

Positions the ToolTip control used by the trackbar control. SetToolTips

void SetToolTips(HWND hwndTip) const;

Assigns a ToolTip control to the trackbar control.

**Base class Members** 

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Like all common controls, the slider control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a slider control.

A slider control is often referred to as a trackbar control.

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes

# **CSocket Class**

## Description

The CSocket class represents a network socket. It encapsualtes many of the Windows Socket SPI fuctions, providing an object-oriented approach to network programming. After StartEvents is called, CSocket monitors the socket and responds automatically to network events. This event monitoring, for example, automatically calls OnReceive when there is data on the socket to be read, and OnAccept when a server should accept a connection from a client.

Users of this class should be aware that functions like OnReceive, OnAccept, etc. are called on a different thread from the one CSocket is instanciated on. The thread for these functions needs to respond quickly to other network events, so it shouldn't be delayed. It also doesn't run a message loop, so it can't be used to create windows. For these reasons it might be best to use PostMessage in response to these functions in a windows environment.

Refer to the network samples for an example of how to use this class to create a TCP client & server, and a UDP client and server.

To compile programs with CSocket, link with ws2\_32.lib for Win32/64, and ws2.lib for Windows CE. Windows 95 systems will need to install the "Windows Sockets 2.0 for Windows 95". It's available from: http://support.microsoft.com/kb/182108/EN-US/

## **CSocket Members:**

#### Construction

	CSocket();
CSocket	Constructs a CSocket object. A CWinException will be thrown if CSocket can't be constructed properly.

### **Operations**

	Accept	<pre>virtual void Accept(CSocket&amp; rClientSock, struct sockaddr* add</pre>
Constructs a CSocket object.		Constructs a CSocket object.
		<pre>virtual int Bind(LPCTSTR addr, UINT port);</pre>
	Bind	

	<pre>virtual int Bind(const struct sockaddr* name, int namelen);</pre>
	Associates a local address with the socket.
	<pre>virtual int Connect(LPCTSTR addr, UINT port);</pre>
Connect	<pre>virtual int Connect(const struct sockaddr* name, int namelen);</pre>
	Establishes a connection to a peer socket.
Create	virtual bool Create( int family, int type, int protocol = IPPR(
	Creates a socket.
Disconnect	virtual void Disconnect();
	Ends any event notification for the socket, shuts the socket down, and removes it from CSock
FreeAddrInfo	virtual void FreeAddrInfo( struct addrinfo* ai );
	Frees address information that the getaddrinfo function dynamically allocates in its addrinfo s
GetAddrInfo	virtual int GetAddrInfo( LPCTSTR nodename, LPCTSTR servname, co
	Provides a protocol-independent translation from host name to address.
ioCtlSocket	<pre>virtual int ioCtlSocket(long cmd, u_long* argp);</pre>
	Controls the I/O mode of the socket.
Listen	virtual int Listen(int backlog = SOMAXCONN);
	Establishes a socket to listen for incoming connection requests.
Receive	virtual int Receive(char* buf, int len, int flags);
	Receives data from the socket.
ReceiveFrom	ReceiveFrom(char* buf, int len, int flags, struct sockaddr* fro
Receiver form	Receives a datagram and stores the source address.
Send	<pre>virtual int Send(LPCTSTR buf, int len, int flags);</pre>
Sella	Sends data to a connected socket.
	virtual int SendTo(const char* send, int len, int flags, LPCTS
SendTo	virtual int SendTo(const char* buf, int len, int flags, const :
	Sends data to a specific destination.
	<pre>virtual void StartEvents();</pre>

	StartEvents	Starts the thread which produces the notification of network events.
	StopEvents	<pre>virtual void StopEvents();</pre>
1		Stops the thread which produces the notification of network events.

## Attributes

GetLastError	<pre>virtual LPCTSTR GetLastError(); Retrieves the calling thread's last-error code value.</pre>		
GetPeerName	<pre>virtual int GetPeerName(struct sockaddr* name, int* namelen) Gets the address of the peer socket connected to socket.</pre>		
GetSocket	SOCKET& GetSocket(); Gets the SOCKET associated with this CSocket.		
GetSockName	<pre>virtual int GetSockName(struct sockaddr* name, int* namelen) Gets the local name of a socket.</pre>		
GetSockOpt	<pre>virtual int GetSockOpt(int level, int optname, char* optval, Gets a socket option.</pre>		
IsIPV6Supported	<pre>virtual bool IsIPV6Supported(); Returns TRUE if the operating system supports IP version 6.</pre>		
SetSockOpt	<pre>virtual int SetSockOpt(int level, int optname, const char* o Sets a socket option.</pre>		

## Overridables

	<pre>virtual void OnAccept();</pre>
OnAccept	Notifies a listening socket that it can accept pending connection requests by calling Accept.
OnAddressListChange	<pre>virtual void OnAddresListChange(); Notifies a socket that there has been a local address list change.</pre>

OnConnect	<pre>virtual void OnDisconnect();</pre>
Cheomeet	Notifies a connecting socket that the connection attempt is complete.
OnDisconnect	<pre>virtual void OnConnect();</pre>
	Notifies a socket that the socket connected to it has closed.
OnOutOfBand	<pre>virtual void OnOutOfBand();</pre>
OllOutOrBallu	Notifies a receiving socket that there is out-of-band data to be read on the socket, usually an urgent message.
OnQualityOfService	<pre>virtual void OnQualityOfService();</pre>
	Notifies a socket that there has been a QOS change.
OnReceive	<pre>virtual void OnReceive();</pre>
	Notifies a listening socket that there is data to be retrieved by calling Receive.
OnRoutingChange	<pre>virtual void OnRoutingChange();</pre>
	Notifies a socket that there has been routing interface change.
OnSend	virtual void OnSend();
	Notifies a socket that it can send data by calling Send.

## Remarks

CSocket's constructor will throw a CWinException if WSAStartup fails or if the WS2\_32.dll DLL fails to load. You can catch this exception if you wish to handle this rather unlikely situation gracefully. Note that windows 95 may not have WS2\_32.dll.

### IP version 6 support

IP version 6 is not supported on all operating systems or all development environments. Keep the following in mind when offering support for IP version 6 in your applications.

- IPv6 is supported on Windows Vista and above. Windows XP with SP2 provides "experimental" support, which can be enabled by entering "ipv6 install" at a command prompt.
- IPv6 is not supported by all compilers and development environments. In particular, it is not supported by Dev-C++ or Borland 5.5. A modern Platform SDK needs to be added to Visual Studio 6

for it to support IPv6.

• The IsIPV6Supported function returns false if either the operating system or the development environment fails to support IPv6.

#### Network client code

The following code segments are complete programs which demonstrate how to write a simple network client and server.

```
#include
#include
#include "winsock2.h"
#include "../../Win32++/socket.h"
using namespace std;
using namespace Win32xx;
class CClientSocket : public CSocket
{
public:
  CClientSocket() {}
  virtual void OnReceive()
  {
    // This function is called automatically when there is data to recei
    TCHAR str[1024] = {0};
    int i = \text{Receive}(\text{str}, 1024, 0);
    cout << i << " chars received: " << str << endl;</pre>
  }
};
int main()
{
  CClientSocket Client;
  // Create the socket to communicate with the Server
  if (!Client.Create(SOCK STREAM))
  {
    cout << "Failed to create socket\n" ;</pre>
    return 0;
  }
  // Connect to the server
  if (SOCKET ERROR == Client.Connect( T("127.0.0.1", 3000)))
  {
    cout << "Failed to connect to server. Was it running?\n";</pre>
    return 0;
```

```
}
cout << "Connected to server.\n";
cout << "Type data to send, type quit to exit\n";
// Monitor the client socket for network events, such as data ready to
Client.StartEvents();
// Send data to the server
string s;
for (;;) // Infinite loop
{
    getline(cin, s);
    if (s == "quit") break;
    int i = Client.Send(s.c_str(), (int)s.length(), 0);
    cout << "Sending " << i << " characters\n";
}
return 0;</pre>
```

Network server code

}

```
#include
#include
#include "winsock2.h"
#include "../../Win32++/socket.h"
using namespace std;
using namespace Win32xx;
class CServerSocket : public CSocket
{
public:
  CServerSocket() {}
 virtual ~CServerSocket() {}
 virtual void OnReceive()
  {
    // This function is called automatically when there is data to recei
    TCHAR str[1024] = {0};
    int i = \text{Receive}(\text{str}, 1024, 0);
    cout << i <<" chars received: " << str << endl;</pre>
 }
};
int main()
{
  // Create the main server socket.
  // It is used to listen for clients
```

```
CServerSocket Server;
 if (!Server.Create(SOCK STREAM))
  {
    cout << "Failed to create socket\n" ;</pre>
    return 0;
 }
 // Bind the IP address and port# to the main socket
 if (SOCKET ERROR == Server.Bind( T("127.0.0.1", 3000)))
 {
    cout << "Failed to bind IP address to socket\n" ;</pre>
    return 0;
 }
 // Listen on the socket for clients to connect
 if (SOCKET ERROR == Server.Listen())
 {
    cout << "Listen on socket failed\n";</pre>
    return 0;
 }
 // Create the socket to communicate with the Client
 CServerSocket Client;
 cout << "Waiting for the client to connect\n";</pre>
 do
  {
   Server.Accept(Client, NULL, NULL);
 while (SOCKET ERROR == Client.GetSocket());
 cout << "Client connected\n";</pre>
 // Monitor the client socket for network events, such as data ready to
 Client.StartEvents();
 // Send data to the client
 cout << "Type data to send, type quit to exit\n";</pre>
 string s;
 for (;;) // infinite loop
  {
    getline(cin, s);
    if (s == "quit") break;
   int i = Client.Send(s.c_str(), (int)s.length(), 0);
    cout << "Sending " << i << " characters\n";</pre>
 }
  return 0;
}
```

Refer to the Networking ClientDlg and ServerDlg sample for a more comprehensive demonstration of the features of CSocket.

Header file	socket.h
Win32/64 support	Yes
WinCE support	Yes
Library required	ws2_32.lib, or ws2.lib for WinCE

# **CSpinButton Class**

## Description

The CSpinButton class is used to create and manage a spin button control.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of spin button controls.

## **CSpinButton Members**

# CSpinButton CSpinButton(); Constructor for the CSpinButton. GetAccel int GetAccel(int cAccels, LPUDACCEL paAccels) const; Retrieves acceleration information for the up-down control. GetBase

int GetBase() const;

Retrieves the current radix base (either base 10 or 16) for the up-down control. GetBuddy

HWND GetBuddy() const;

Retrieves the handle to the current buddy window. GetPos

int GetPos() const;

Retrieves the current position of the upd-down control with 16-bit precision. GetRange

DWORD GetRange() const;

Retrieves the minimum and maximum positions (range) for the up-down control. SetAccel

BOOL SetAccel(int cAccels, LPUDACCEL paAccels) const;

Sets the acceleration for the up-down control. SetBase

#### int SetBase(int nBase) const;

Sets the radix base (either 10 or 16) for the up-down control. SetBuddy

```
HWND SetBuddy(HWND hwndBuddy) const;
```

Sets the buddy window for the up-down control. SetPos

```
int SetPos(int nBase) const;
```

Sets the current position for the up-down control with 16-bit precision. SetRange

```
void SetRange(int nLower, int nUpper) const;
```

Sets the minimum and maximum positions (range) for the up-down control.

### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Like all common controls, the spin button control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a spin button control.

A spin button control is often referred to as an up-down control.

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes

# **CStatic Class**

## Description

Applications often use static controls to label other controls or to separate a group of controls. Although static controls are child windows, they cannot be selected. Therefore, they cannot receive the keyboard focus and cannot have a keyboard interface. A static control that has the SS\_NOTIFY style receives mouse input, notifying the parent window when the user clicks or double clicks the control. Static controls belong to the STATIC window class.

CStatic is the class responsible for creating a static control.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of static controls.

## **CStatic Members**

#### Construction

CStatic	
	CStatic();
	Constructs a CStatic

### Attributes

GetBitmap	
	HBITMAP GetBitmap() const;
	Returns the handle of the bitmap associated with the static control set by
	CStatic::SetBitmap.
GetCursor	HCURSOR GetCursor() const;
	Returns the handle of the cursor associated with the static control set by
	CStatic::SetCursor.
	HENHMETAFILE GetEnhMetaFile() const;
GetEnhMetaFile	
	Returns the handle of the enhanced metafile associated with the static control set by
	CStatic::SetEnhMetaFile.
Catlana	
GetIcon	HICON GetIcon() const;

	Returns a handle to the icon associated with the static control set by CStatic::SetIcon.
SetBitmap	HBITMAP SetBitmap(HBITMAP hBitmap) const;
	Associates a new bitmap with the static control. The bitmap will be drawn in the upper- left corner. The static control will be resized to the size of the bitmap. This function requires the SS_BITMAP style.
SetCursor	HCURSOR SetCursor(HCURSOR hCursor) const;
	Associates a new cursor image with static control. The cursor will be drawn in the upper- left corner. The static control will be resized to the size of the cursor. This function requires the SS_ICON style.
	<pre>HENHMETAFILE SetEnhMetaFile(HENHMETAFILE hMetaFile) const;</pre>
SetEnhMetaFile	
SetIcon	UTCON Cottoon (UTCON hToon) const.
	HICON SetIcon(HICON hIcon) const; Associates a new icon image with the static control. The icon will be drawn in the upper left of the static control. The static control will be resized to the size of the icon. This function requires the SS_ICON style.

### **Overidables**

	<pre>virtual void PreRegisterClass(WNDCLASS &amp;wc);</pre>
	Set the window class parameters before the window is created.

### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Like all common controls, the static control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a static control.

Although static controls can be used in overlapped, pop-up, and child windows, they are designed for use in dialog boxes, where the system standardizes their behavior. By using static controls outside dialog boxes, a developer increases the

risk that the application might behave in a nonstandard fashion. Typically, a developer either uses static controls in dialog boxes or uses the SS\_OWNERDRAW style to create customized static controls.

Header file	stdcontrols.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CString Class**

## Description

This class is intended to provide much the same functionality of the MFC/ATL CString class that ships with Microsoft compilers. The CString class specified here is compatible with other compilers such as Borland 5.5 and MinGW.

## **CString Members**

Initialisation and Assignment		
	<pre>CString();</pre>	
	CString(LPCSTR pszText);	
CString	CString(LPCWSTR pszText);	
0	CString(TCHAR ch, int nLength = 1);	
	CString(LPCTSTR pszText, int nLength);	
	Constructor for CString.	
operator =		
CString& operator = (const CString& str);		
CString& operator = (const TCHAR ch);		
CString& operator = (LPCSTR pszText);		
CString& operator = (LPCWSTR pszText);		
Assigns a value to the CString. <b>Attributes</b> c_str		
LPCTSTR c_str() const;		
Converts the CString to a LPCTSTR. GetString		

tString& GetString();

Returns a reference to the underlying std::basic\_string<TCHAR>. GetLength

int GetLength() const;

Returns the length in characters. **Operator LPCTSTR** 

operator LPCTSTR() const;

Converts the CString to a LPCTSTR. Operator BSTR

operator BSTR() const;

Converts the CString to a BSTR. **Operations** AllocSysString

BSTR AllocSysString() const;

Allocates a BSTR from the CString content. AppendFormat

void AppendFormat(LPCTSTR pszFormat,...);

void AppendFormat(UINT nFormatID, ...);

Appends formatted data to an the CString content. Collate

int Collate(LPCTSTR pszText) const;

Performs a case sensitive comparison of the two strings using locale-specific information. CollateNoCase

int CollateNoCase(LPCTSTR pszText) const;

Performs a case insensitive comparison of the two strings using locale-specific information.

### Compare

int Compare(LPCTSTR pszText) const;

Performs a case sensitive comparison of the two strings. CompareNoCase

int CompareNoCase(LPCTSTR pszText) const;

Performs a case insensitive comparison of the two strings. Delete

int Delete(int nIndex, int nCount = 1);

Deletes a character or characters from the string. Empty

void Empty();

Erases the contents of the string. Find

int Find(TCHAR ch, int nIndex = 0 ) const;

int Find(LPCTSTR pszText, int nStart = 0) const;

Finds a character in the string. FindOneOf

int FindOneOf(LPCTSTR pszText) const;

Finds the first matching character from a set. Format

void Format(UINT nID, ...);

void Format(LPCTSTR pszFormat,...);

Formats the string as sprintf does. FormatV

void FormatV(LPCTSTR pszFormat, va\_list args);

Formats the string using a variable list of arguments. FormatMessage

void FormatMessage(LPCTSTR pszFormat,...);

Formats a message string. FormatMessageV

void FormatMessageV(LPCTSTR pszFormat, va\_list args);

Formats a message string using a variable argument list. GetAt

TCHAR GetAt(int nIndex) const;

Returns the character at the specified location within the string. GetBuffer

### LPTSTR GetBuffer(int nMinBufLength);

Creates a buffer of nMinBufLength charaters (+1 extra for NULL termination) and returns a pointer to this buffer. This buffer can be used by any function which accepts a LPTSTR. Care must be taken not to exceed the length of the buffer. Use ReleaseBuffer to copy this buffer back to the CString object.

## GetEnvironmentVariable

BOOL GetEnvironmentVariable(LPCTSTR pszVar);

Sets the string to the value of the specified environment variable. GetErrorString

#### void GetErrorString(DWORD dwError);

Returns the error string for the specified System Error Code (e.g from GetLastError). Insert

int Insert(int nIndex, TCHAR ch);

int Insert(int nIndex, const CString& str);

Inserts a single character or a substring at the given index within the string. Is Empty

BOOL IsEmpty() const;

Returns TRUE if the string is empty. Left

CString Left(int nCount) const;

Extracts the left part of a string. LoadString

```
BOOL LoadString(UINT nID);
```

Loads the string from a Windows resource. MakeLower

void MakeLower();

Converts all the characters in this string to lowercase characters. MakeReverse

void MakeReverse();

Reverses the string. MakeUpper

void MakeUpper();

Converts all the characters in this string to uppercase characters.  $\operatorname{Mid}$ 

CString Mid(int nFirst) const;

CString Mid(int nFirst, int nCount) const;

Extracts the middle part of a string. ReleaseBuffer

void ReleaseBuffer( int nNewLength = -1 );

This copies the contents of the buffer (acquired by GetBuffer) to this CString, and releases the contents of the buffer. The default length of -1 copies from the buffer until a null terminator is reached. If the buffer

doesn't contain a null terminator, you must specify the buffer's length. Remove

int Remove(LPCTSTR pszText);

Removes each occurrence of the specified substring from the string. Replace

int Replace(TCHAR ch0ld, TCHAR chNew);

int Replace(const LPCTSTR psz0ld, LPCTSTR pszNew);

Replaces each occurance of the old character with the new character. **ReverseFind** 

int ReverseFind(LPCTSTR pszText, int nStart = -1) const;

Search for a substring within the string, starting from the end. Right

CString Right(int nCount) const;

Extracts the right part of a string. SetAt

void SetAt(int nIndex, TCHAR ch);

Sets the character at the specified position to the specified value. SetSysString

BSTR SetSysString(BSTR\* pBstr) const;

Sets an existing BSTR object to the string. SpanExcluding

CString SpanExcluding(LPCTSTR pszText) const;

Extracts characters from the string, starting with the first character, that are not in the set of characters identified by pszCharSet. **SpanIncluding** 

CString SpanIncluding(LPCTSTR pszText) const;

Extracts a substring that contains only the characters in a set. Tokenize

CString Tokenize(LPCTSTR pszTokens, int& iStart) const;

Extracts specified tokens in a target string. Trim

void Trim();

Trims all leading and trailing whitespace characters from the string. TrimLeft

void TrimLeft();

void TrimLeft(TCHAR chTarget);

void TrimLeft(LPCTSTR pszTargets);

# Trims white space characters or the specified set of characters from the beginning of the string. TrimRight

TrimRight();

void TrimRight(TCHAR chTarget);

void TrimRight(LPCTSTR pszTargets);

Trims trailing whitespace characters or the specified set of characters from the string. Truncate

void Truncate(int nNewLength);

Reduces the length of the string to the specified amount. **Operator** +

friend CString operator + (const CString& string1, const CString& string

friend CString operator + (const CString& string, LPCTSTR pszText);

friend CString operator + (const CString& string, TCHAR ch);

friend CString operator + (LPCTSTR pszText, const CString& string);

friend CString operator + (TCHAR ch, const CString& string);

Appends the specified characters to the string. **Operator** +=

CString& operator += (const CString& str);

CString& operator += (LPCSTR szText);

CString& operator += (LPCWSTR szText);

CString& operator += (const TCHAR ch);

Appends and then assigns the specified characters to the string. **Operator** 

TCHAR& operator [] (int nIndex);

Returns the character at the specified index. **Comparisons** operator ==

bool operator == (LPCTSTR pszText) const;

Performs a case sensitive comparison of the two strings. Returns true of the two strings are the same. **Operator** !=

### bool operator != (LPCTSTR pszText) const;

Performs a case sensitive comparison of the two strings. Returns true if the two strings are not the same.

### operator <

friend bool operator < (const CString& string1, const CString& string2);
friend bool operator < (const CString& string1, LPCTSTR pszText);</pre>

Performs a case sensitive comparison of the two strings. **Operator** >

friend bool operator > (const CString& string1, const CString& string2);

friend bool operator > (const CString& string1, LPCTSTR pszText);

Performs a case sensitive comparison of the two strings. **Operator**  $\leq$  =

friend bool operator <= (const CString& string1, const CString& string2)</pre>

friend bool operator <= (const CString& string1, LPCTSTR pszText);</pre>

Performs a case sensitive comparison of the two strings. **Operator**  $\geq$  =

friend bool operator >= (const CString& string1, const CString& string2)

friend bool operator >= (const CString& string1, LPCTSTR pszText);

Performs a case sensitive comparison of the two strings.

### Remarks

#### Using the CString class

The following code demonstrates how to assign strings to a CString.

```
// Assign some text to CString
CString str1 = _T("Hello World.");
CString str2 = _T(" I like strings.");
CString str3 = str1 + str2;
SetWindowText(str3);
```

```
// How to use GetBuffer and SetBuffer with functions expecting a TCHAR a
CString str;
int nLength = ::GetWindowTextLength(m_hWnd);
::GetWindowText(m_hWnd, str.GetBuffer(nLength), nLength + 1);
str.ReleaseBuffer();
```

### Differences between this class and the MFC/ATL CString class

- The constructors for this class accepts only TCHARs. The various text conversion functions can be used to convert from other character types to TCHARs.
- This class is not reference counted, so these CStrings should be passed as references or const references when used as function arguments. As a result there

is no need for functions like LockBuffer and UnLockBuffer.

• The Format functions only accepts POD (Plain Old Data) arguments. It does not accept arguments which are class or struct objects. In particular it does not accept CString objects, unless these are cast to LPCTSTR. This is demonstrates valid and invalid usage:

```
CString string1(_T("Hello World"));
CString string2;
// This is invalid, and produces undefined behaviour.
string2.Format(_T("String1 is: %s"), string1); // No! you can't do t
// This is ok
string2.Format(_T("String1 is: %s"), (LPCTSTR)string1); // Yes, this
```

- This class provides a few additional functions:
  - c\_str Returns a const TCHAR string. This is an alternative for casting to LPCTSTR.
  - GetErrorString Assigns CString to the error string for the specified System Error Code.
  - GetString Returns a reference to the underlying std::basic\_string<TCHAR>.

Header file	c_string.h	
Win32/64	Yes	
support		
WinCE support	Yes	

## **CTab Class**

### Description

A tab control is window that displays one or more tabs, and a window page. It is somewhat analogous to the dividers in a notebook or the labels in a file cabinet. By using a tab control, an application can define multiple pages for the same area of a window or dialog box.

CTab is the class which creates a tab control. The tabs typically have an icon, and some text. The tabs can be displayed either at the top or bottom of the window. Additional buttons to allow the tabs to be listed or closed can also be displayed.

The AddTabPage function is used to add new tabs to the control. This function provides an opportunity to specify the tab's icon and text, as well as the window which will be used as this tab's page.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of tab controls.

#### **CTab Members**

#### Operations

CTab	CTab();	
	Constructor for CTab	
AddTabP	age	
virtual int	: AddTabPage(ViewPtr pView, LPCTSTR szTabText, HICON hIcon, UI	
virtual int	AddTabPage(ViewPtr pView, LPCTSTR szTabText, UINT nID_Icon,	
virtual int	: AddTabPage(ViewPtr pView, LPCTSTR szTabText);	
Adds a tab page from the specified view window, tab text and tab icon. $\operatorname{AdjustRect}$		
void Adjust	Rect(BOOL fLarger, RECT *prc) const;	
Calculates a tab control's display area given a window rectangle, or calculates the window rectangle that would correspond to a specified display area. $DeleteAllItems$		
BOOL Delete	AllItems() const;	

Removes all items from a tab control. DeleteItem

B00L DeleteItem(int iItem) const;

Removes an item from a tab control. DeselectAll

void DeselectAll(UINT fExcludeFocus) const;

Resets items in a tab control, clearing any that were set to the TCIS\_BUTTONPRESSED state. GetActiveView

CWnd\* GetActiveView() const;

Returns a pointer to the currently active view window GetAllTabs

std::vector <TabPageInfo>\* GetAllTabs() const;

Returns a pointer to the vector of tab page information for all tabs. GetCloseRect

virtual CRect GetCloseRect() const;

Returns the dimensions of the bounding rectangle of the close button. GetCurFocus

int GetCurFocus() const;

Returns the index of the item that has the focus in a tab control. GetCurSel

int GetCurSel() const;

Determines the currently selected tab in a tab control. GetExtendedStyle

DWORD GetExtendedStyle() const;

Retrieves the extended styles that are currently in use for the tab control. GetImageList

CImageList\* GetImageList()const;

Returns a pointer to the tab control's image list. GetItem

BOOL GetItem(int iItem, LPTCITEM pitem) const;

Retrieves information about a tab in a tab control. GetItemCount

int GetItemCount() const;

Retrieves the number of tabs in the tab control. GetItemRect

BOOL GetItemRect(int iItem, LPRECT prc) const;

Retrieves the bounding rectangle for the specified tab in a tab control. GetListMenu

virtual CMenu\* GetListMenu();

Returns a pointer to the list menu. GetODImageList

CImageList\* GetODImageList() const

Returns the image list for owner drawn tabs. GetListRect

virtual CRect GetListRect() const;

Returns the dimensions of the bounding rectangle of the list button. GetRowCount

int GetRowCount() const;

Retrieves the current number of rows of tabs in a tab control. GetShowButtons

B00L GetShowButtons() const;

Returns TRUE if the list and close buttons are displayed. GetTabFont

CFont\* GetTabFont() const;

Returns the CFont used to set the tab's font. GetTabHeight

int GetTabHeight() const;

Returns the height of the tabs. GetTabImageID

virtual int GetTabImageID(UINT nTab) const;

Returns the image ID for the specified tab. GetTabIndex

virtual int GetTabIndex(CWnd\* pWnd) const;

Returns the index of the tab given its view window. GetTabPageInfo

virtual TabPageInfo GetTabPageInfo(UINT nTab) const;

Returns the tab page info struct for the specified tab. GetTabText

#### virtual CString GetTabText(UINT nTab) const;

Returns the text for the specified tab. GetTabsAtTop

virtual BOOL GetTabsAtTop() const;

Returns TRUE if the tabs are displayed at the top of the control, and FALSE if they are displayed at the bottom. GetToolTips

CToolTip\* GetToolTips() const;

Retrieves a pointer to the ToolTip control associated with a tab control. HighlightItem

BOOL HighlightItem(INT idItem, WORD fHighlight) const;

Sets the highlight state of a tab item. HitTest

int HitTest(TCHITTESTINF0& info) const;

Determines which tab, if any, is at a specified screen position. InsertItem

int InsertItem(int iItem, const LPTCITEM pItem) const;

Inserts a new tab in a tab control. **RecalcLayout** 

virtual void RecalcLayout();

Repositions the child windows of the tab control. RemoveImage

void RemoveImage(int iImage) const;

Removes an image from a tab control's image list. RemoveTabPage

virtual void RemoveTabPage(int nPage);

Removes a tab page from the control. SelectPage

virtual void SelectPage(int nPage);

Makes the specified tab index the selected tab page. SetCurFocus

void SetCurFocus(int iItem) const;

Sets the focus to a specified tab in a tab control. SetCurSel

int SetCurSel(int iItem) const;

Selects a tab in a tab control. SetExtendedStyle

DWORD SetExtendedStyle(DWORD dwExStyle) const;

Sets the extended styles that the tab control will use. SetFixedWidth

virtual void SetFixedWidth(BOOL bEnabled);

Enable or disable fixed tab width. SetFont

virtual void SetFont(CFont\* pFont, BOOL bRedraw = 1);

Sets the font and adjusts the tab height to match. SetImageList

CImageList\* SetImageList(CImageList\* pImageList) const;

Assigns an image list to a tab control. SetItem

BOOL SetItem(int iItem, LPTCITEM pitem) const;

Sets some or all of a tab's attributes. SetItemExtra

BOOL SetItemExtra(int cb) const;

Sets the number of bytes per tab reserved for application-defined data in a tab control. SetItemSize

DWORD SetItemSize(int cx, int cy) const;

Sets the width and height of tabs. SetMinTabWidth

int SetMinTabWidth(int cx) const;

Sets the minimum width of items in a tab control. SetOwnerDraw

virtual void SetOwnerDraw(BOOL bEnabled);

Enable or disable owner draw. Set owner draw to TRUE to display the tab icons. SetPadding

void SetPadding(int cx, int cy) const;

Sets the amount of space (padding) around each tab's icon and label in a tab control. SetShowButtons

virtual void SetShowButtons(BOOL bShow);

Allows the list and close buttons to be shown or hidden. SetTabHeight

void SetTabHeight(int nTabHeight);

Sets the height of the tabs. **SetTabIcon** 

```
virtual void SetTabIcon(int i, HICON hIcon);
```

Change the tab's existing icon, or assigns a new icon if one didn't previously exist. SetTabsAtTop

virtual void SetTabsAtTop(BOOL bTop);

Positions the tabs at the top or bottom of the control.  $\ensuremath{SetTabText}$ 

virtual void SetTabText(UINT nTab, LPCTSTR szText);

Sets the text of the specified tab SwapTabs

virtual void SwapTabs(UINT nTab1, UINT nTab2);

Swaps the two specified tabs. SetToopTipS

void SetToolTips(CToolTip\* pToolTip) const;

Assigns a ToolTip control to a tab control.

#### **Overridables**

DrawCloseButton	<pre>virtual void DrawCloseButton(CDC&amp; DrawDC); Draws the close button.</pre>
DrawListButton	<pre>virtual void DrawListButton(CDC&amp; DrawDC); Draws the list button.</pre>
DrawTabs	<pre>virtual void DrawTabs(CDC&amp; dcMem); Draws the tabs.</pre>
DrawTabBorders	<pre>virtual void DrawTabBorders(CDC&amp; dcMem, CRect&amp; rcTab); Draws the tab borders.</pre>
OnCreate	<pre>virtual void OnCreate();</pre>

	Called when the tab control is created.
OnEraseBkgnd	virtual LRESULT OnEraseBkgnd(WPARAM wParam, LPARAM lParam)
	Called when the tab control's background is redrawn.
OnKillFocus	<pre>virtual LRESULT OnKillFocus(WPARAM wParam, LPARAM lParam);</pre>
	Called when the tab control loses focus.
OnLButtonDblClk	virtual LRESULT OnLButtonDblClk(WPARAM wParam, LPARAM lPara
	Called when the left mouse button is double clicked.
OnLButtonDown	virtual LRESULT OnLButtonDown(WPARAM wParam, LPARAM lParam
OILDuttoIIDOWII	Called when the left mouse button is clicked.
OnLButtonUp	<pre>virtual void OnLButtonUp(WPARAM wParam, LPARAM lParam);</pre>
Опграцопор	Called when the left mouse button is released.
O.M.	virtual void OnMouseLeave(WPARAM wParam, LPARAM lParam);
OnMouseLeave	Called when the mouse leaves the tab control.
	virtual void OnMouseMove(WPARAM wParam, LPARAM lParam);
OnMouseMove	
	Called when the mouse moves over the tab control.
OnNCHitTest	Called when the mouse moves over the tab control. virtual LRESULT OnNCHitTest(WPARAM wParam, LPARAM lParam); Called when the mouse moves, or when a mouse button is pressed or released over the t (non-client hit test).
	<pre>virtual LRESULT OnNCHitTest(WPARAM wParam, LPARAM lParam); Called when the mouse moves, or when a mouse button is pressed or released over the t</pre>
OnNCHitTest OnNotifyReflect	<pre>virtual LRESULT OnNCHitTest(WPARAM wParam, LPARAM lParam); Called when the mouse moves, or when a mouse button is pressed or released over the t (non-client hit test).</pre>
	<pre>virtual LRESULT OnNCHitTest(WPARAM wParam, LPARAM lParam); Called when the mouse moves, or when a mouse button is pressed or released over the t (non-client hit test). virtual LRESULT OnNotifyReflect(WPARAM wParam, LPARAM lPara</pre>
	<pre>virtual LRESULT OnNCHitTest(WPARAM wParam, LPARAM lParam); Called when the mouse moves, or when a mouse button is pressed or released over the t (non-client hit test). virtual LRESULT OnNotifyReflect(WPARAM wParam, LPARAM lPara Called when the tab control sends a notification.</pre>
OnNotifyReflect OnSetFocus	<pre>virtual LRESULT OnNCHitTest(WPARAM wParam, LPARAM lParam); Called when the mouse moves, or when a mouse button is pressed or released over the t (non-client hit test). virtual LRESULT OnNotifyReflect(WPARAM wParam, LPARAM lPara Called when the tab control sends a notification. virtual LRESULT OnSetFocus(WPARAM wParam, LPARAM lParam); Called when the tab control gets keyboard focus. We set the keyboard focus to the activ</pre>
OnNotifyReflect	<pre>virtual LRESULT OnNCHitTest(WPARAM wParam, LPARAM lParam); Called when the mouse moves, or when a mouse button is pressed or released over the t (non-client hit test). virtual LRESULT OnNotifyReflect(WPARAM wParam, LPARAM lPara Called when the tab control sends a notification. virtual LRESULT OnSetFocus(WPARAM wParam, LPARAM lParam); Called when the tab control gets keyboard focus. We set the keyboard focus to the activ window.</pre>
OnNotifyReflect OnSetFocus OnTCNSelChange	<pre>virtual LRESULT OnNCHitTest(WPARAM wParam, LPARAM lParam); Called when the mouse moves, or when a mouse button is pressed or released over the t (non-client hit test). virtual LRESULT OnNotifyReflect(WPARAM wParam, LPARAM lPara Called when the tab control sends a notification. virtual LRESULT OnSetFocus(WPARAM wParam, LPARAM lParam); Called when the tab control gets keyboard focus. We set the keyboard focus to the activ window. virtual LRESULT OnTCNSelChange(LPNMHDR pNMHDR);</pre>
OnNotifyReflect OnSetFocus	<pre>virtual LRESULT OnNCHitTest(WPARAM wParam, LPARAM lParam); Called when the mouse moves, or when a mouse button is pressed or released over the t (non-client hit test). virtual LRESULT OnNotifyReflect(WPARAM wParam, LPARAM lPar; Called when the tab control sends a notification. virtual LRESULT OnSetFocus(WPARAM wParam, LPARAM lParam); Called when the tab control gets keyboard focus. We set the keyboard focus to the activ window. virtual LRESULT OnTCNSelChange(LPNMHDR pNMHDR); Called when the currently selected tab has changed</pre>

OnWindowPosChanging	virtual LRESULT OnWindowPosChanging(WPARAM wParam, LPARAM
	Called when the window's position is changing.
Paint	virtual void Paint();
	Paints the tab control.
PreCreate	<pre>virtual void PreCreate(CREATESTRUCT&amp; cs);</pre>
	Specifies the window creation parameters before the window is created.
PreRegisterClass	<pre>virtual void PreRegisterClass(WNDCLASS &amp;wc);</pre>
	Set the window class parameters before the window is created
setTabSize	<pre>virtual void SetTabSize();</pre>
Sectubolize	Sets the size of the tabs.
ShowListDialog	<pre>virtual void ShowListDialog();</pre>
	Displays the list of windows in a dialog.
ShowListMenu	<pre>virtual void ShowListMenu();</pre>
	Displays the list of windows in a popup menu.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

#### Remarks

Like all common controls, the tab control requires a parent window. This parent window is often a <u>dialog</u>, but other windows, such as <u>simple windows</u> or <u>Frames</u>, can also be the parent window for a tab control.

The following example shows how to add pages to a tab control.

```
void CMainFrame::OnInitialUpdate()
{
    // Add some tabs to the tab control
    SetOwnerDraw(TRUE);
    SetFixedWidth(TRUE);
    m_View.AddTabPage(new CViewClasses, _T("Classes"), IDI_CLASSVIEW);
    m_View.AddTabPage(new CViewFiles, _T("Files"), IDI_FILEVIEW);
    m_View.AddTabPage(new CViewClasses, _T("Classes"), IDI_CLASSVIEW);
```

```
m_View.AddTabPage(new CViewFiles, _T("Files"), IDI_FILEVIEW);
m_View.SelectPage(0);
```

SetOwnerDraw disables or enables owner drawing for the tab control. When owner drawing is enabled, flickering in the tab's view window is reduced and tabs at the bottom of the tab control are drawn properly. When owner drawing is disabled, the tab control reverts back to its default drawing and tab rendering.

Refer to the TabDemo for a demonstration of using CTab as the view window within a Frame, and TabDialogDemo for a demonstration of using CTab in a dialog.

Other Win32++ class that use Tab controls include <u>CTabbedMDI</u> and <u>CDockContainer</u>. <u>CTabbedMDI</u> combines many of the features of a Tab Control and a <u>MDI Frame</u>. <u>CDockContainer</u> is a specialized Tab control intend for use with <u>Dockers</u>.

### **Summary Information**

}

Header file	tab.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CTabbedMDI** Class

## Description

CTabbedMDI class combines many of the features of a MDI Frame and a tab control. Each MDI child is displayed as a separate tab page. Buttons which allow the MDI children to be listed or closed are displayed to the right of the tabs. The tabs can be displayed either at the top or bottom of the window.

## **CTabbedMDI** Members

#### Operations

CTabbedMDI();
Constructor for CTabbedMDI
AddMDIChild
<pre>virtual CWnd* AddMDIChild(WndPtr pView, LPCTSTR szTabText, int idMDIChi</pre>
Adds a MDI tab, given a pointer to the view window, and the tab's text. $CloseActiveMDI$
<pre>virtual void CloseActiveMDI();</pre>
Closes the active MDI child CloseAllMDIChildren
<pre>virtual void CloseAllMDIChildren();</pre>
Closes all MDI children CloseMDIChild
<pre>virtual void CloseMDIChild(int nTab);</pre>
Closes the MDI child, given the tab's index. GetActiveMDIChild
<pre>virtual CWnd* GetActiveMDIChild() const;</pre>
Retrieves a pointer to the active MDI child's view window. $GetActiveMDITab$
<pre>virtual int GetActiveMDITab() const;</pre>

Retrieves the index of the current active MDI tab. GetListMenu

virtual HMENU GetListMenu() const

Retrieves a pointer to the MDI child's view window, give the tab's index. GetMDIChild

virtual CWnd\* GetMDIChild(int nTab) const;

Retrieves a pointer to the MDI child's view window, give the tab's index. GetMDIChildCount

virtual int GetMDIChildCount() const;

Retrieves the number of MDI children. GetMDIChildID

virtual int GetMDIChildID(int nTab) const;

Retrieves the MDI child's ID, given the tab's index. GetMDIChildTitle

virtual LPCTSTR GetMDIChildTitle(int nTab) const;

Retrieves the MDI child's text, given the tab's index GetTab

virtual CTab\* GetTab() const;

Returns a pointer to the TabbedMDI's tab control. LoadRegistrySettings

virtual BOOL LoadRegistrySettings(CString strRegistryKeyName);

Loads the TabbedMDI information from the registry. **RecalcLayout** 

virtual void RecalcLayout();

Repositions the child windows of the TabbedMDI window. SaveRegistrySettings

virtual BOOL SaveRegistrySettings(CString strRegistryKeyName);

Saves the TabbedMDI information in the registry SetActiveMDIChild

virtual void SetActiveMDIChild(CWnd\* pWnd);

Sets the active MDI child, given its view window. SetActiveMDITab

virtual void SetActiveMDITab(int nTab);

Sets the active MDI child given its tab index.

#### **Overridables**

NewMDIChildFromID	<pre>virtual CWnd* NewMDIChildFromID(int idMDIChild);</pre>
	Override this function to create a new Docker given its TabbedMDI ID.
OnCreate	<pre>virtual void OnCreate() ;</pre>
	Called when the TabbedMDI window is created.
OnDestroy	<pre>virtual void OnDestroy(WPARAM wParam, LPARAM lParam);</pre>
Chillesticy	Called when the TabbedMDI window is destroyed.
OnNotify	<pre>virtual LRESULT OnNotify(WPARAM wParam, LPARAM lParam);</pre>
	Called when the TabbedMDI recieves a notification.
OnWindowPosChanged	virtual void OnWindowPosChanged(WPARAM wParam, LPARAM lf
	Called when the TabbedMDI window is resized or repositioned.

#### Remarks

The following example demonstrates how to toggle the MDI tab position between the top and bottom of the window.

```
void CMainFrame::OnMDITabsAtTop()
// Toggle the TabbedMDI's tabs between the top to bottom of window
{
    CTabbedMDI* pTabbedMDI = m_DockTabbedMDI.GetTabbedMDI();
    BOOL bTop = pTabbedMDI->GetTab().GetTabsAtTop();
    pTabbedMDI->GetTabsAtTop(!bTop);
    // Set the menu checkmark
    UINT uCheck = (bTop)? MF_UNCHECKED : MF_CHECKED;
    ::CheckMenuItem(GetFrameMenu(), IDM_TABBEDMDI_TOP, uCheck);
}
```

The following example demonstrates how to add MDI children to the TabbedMDI control.

```
void CMainFrame::LoadDefaultMDIs()
{
    // Add some MDI tabs
```

```
CTabbedMDI* pTabbedMDI = (CTabbedMDI*)m_DockTabbedMDI.GetView();
pTabbedMDI->AddMDIChild(new CViewSimple, _T("Simple View"), ID_MDI_SIM
pTabbedMDI->AddMDIChild(new CViewRect, _T("Rectangles"), ID_MDI_RECT);
pTabbedMDI->AddMDIChild(new CViewText, _T("TextView"), ID_MDI_TEXT);
pTabbedMDI->AddMDIChild(new CViewClasses, _T("Classes"), ID_MDI_CLASSE
pTabbedMDI->AddMDIChild(new CViewFiles, _T("Files"), ID_MDI_FILES);
pTabbedMDI->SetActiveMDITab(0);
}
```

**Base class Members** 

For base class members, refer to the members of <u>CWnd</u>.

#### Remarks

Refer to the DockTabbedMDI sample for a demonstration of the use of CTabbedMDI.

### **Summary Information**

Header file	tab.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

# **CTaskDialog Class**

## Description

The CTaskDialog class is used to create Task Dialogs. A task dialog is a dialog box that can be used to display information and receive simple input from the user. Like a message box, it is formatted by the operating system according to parameters you set. However, a task dialog has many more features than a message box.

### **CTaskDialog Members**

Construction	
CTaskDialog	

CTaskDialog();

Constructor for the CTaskDialog. Attributes GetConfig

TASKDIALOGCONFIG GetConfig() const;

Returns the TASKDIALOGCONFIG structure for the Task Dialog. GetOptions

TASKDIALOG\_FLAGS GetOptions() const;

Returns the Task Dialog's options. GetSelectedButtonID

int GetSelectedButtonID() const;

Returns the ID of the selected button. GetSelectedRadioButtonID

int GetSelectedRadioButtonID() const;

Returns the ID of the selected radio button. GetVerificationCheckboxState

B00L GetVerificationCheckboxState() const;

Returns the state of the verification check box. IsSupported

static BOOL IsSupported();

Returns true if TaskDialogs are supported on this system. SetCommonButtons

void SetCommonButtons(TASKDIALOG\_COMMON\_BUTTON\_FLAGS dwCommonButtons);

Sets the common buttons for the task dialog. SetContent

void SetContent(LPCTSTR pszContent);

Sets the task dialog's primary content. SetDefaultButton

void SetDefaultButton(int nButtonID);

Sets the task dialog's default button. SetDefaultRadioButton

void SetDefaultRadioButton(int nRadioButtonID);

Sets the default radio button. SetDialogWidth

void SetDialogWidth(UINT nWidth = 0);

The width of the task dialog's client area. If 0, the task dialog manager will calculate the ideal width. **SetExpansionArea** 

void SetExpansionArea(LPCTSTR pszExpandedInfo, LPCTSTR pszExpandedLabel LPCTSTR pszCollapsedLabel = \_T(""));

Sets the text in the expandable area of the Task Dialog. SetFooterIcon

void SetFooterIcon(HICON hFooterIcon);

void SetFooterIcon(LPCTSTR lpszFooterIcon);

Sets the icon that will be displayed in the Task Dialog's footer. SetFooterText

void SetFooterText(LPCTSTR pszFooter);

Sets the text that will be displayed in the Task Dialog's footer. SetMainIcon

void SetMainIcon(HICON hMainIcon);

void SetMainIcon(LPCTSTR lpszMainIcon);

Sets Task Dialog's main icon. SetMainInstruction

void SetMainInstruction(LPCTSTR pszMainInstruction);

Sets the Task Dialog's main instruction text. SetOptions

void SetOptions(TASKDIALOG\_FLAGS dwFlags);

Sets the Task Dialog's options. SetProgressBarMarquee

void SetProgressBarMarquee(BOOL bEnabled = TRUE, int nMarqueeSpeed = 0);

Starts and stops the marquee display of the progress bar, and sets the speed of the marquee. SetProgressBarPosition

void SetProgressBarPosition(int nProgressPos);

Sets the current position for a progress bar. SetProgressBarRange

void SetProgressBarRange(int nMinRange, int nMaxRange);

Sets the minimum and maximum values for the hosted progress bar. SetProgressBarState

void SetProgressBarState(int nNewState = PBST\_NORMAL);

Sets the current state of the progress bar. SetVerificationCheckbox

void SetVerificationCheckbox(BOOL bChecked);

Simulates a click on the verification checkbox of the Task Dialog, if it exists. **SetVerificationCheckboxText** 

void SetVerificationCheckboxText(LPCTSTR pszVerificationText);

Sets the text for the verification check box. SetWindowTitle

void SetWindowTitle(LPCTSTR pszWindowTitle);

Sets the Task Dialog's window title. **Operations** AddCommandControl

void AddCommandControl(int nButtonID, LPCTSTR pszCaption);

Adds a command control or push button to the Task Dialog. AddRadioButton

void AddRadioButton(int nRadioButtonID, LPCTSTR pszCaption);

Adds a radio button to the Task Dialog. AddRadioButtonGroup

void AddRadioButtonGroup(int nIDRadioButtonsFirst, int nIDRadioButtonsLa

# Adds a range of radio buttons to the Task Dialog. The resource ID of the button and it's string must match. **ClickButton**

#### void ClickButton(int nButtonID) const;

Simulates the action of a button click in the Task Dialog. ClickRadioButton

void ClickRadioButton(int nRadioButtonID) const;

Simulates the action of a radio button click in the TaskDialog. DoModal

LRESULT DoModal(CWnd\* pParent = NULL);

Creates and displays the Task Dialog. ElevateButton

void ElevateButton(int nButtonID, BOOL bElevated);

Adds a shield icon to indicate that the button's action requires elevated privilages. EnableButton

void EnableButton(int nButtonID, BOOL bEnabled);

Enables or disables a push button in the TaskDialog. EnableRadioButton

void EnableRadioButton(int nButtonID, BOOL bEnabled);

Enables or disables a radio button in the TaskDialog. NavigateTo

void NavigateTo(CTaskDialog& TaskDialog) const;

Replaces the information displayed by the task dialog. RemoveAllButtons

void RemoveAllButtons();

Removes all push buttons from the task dialog. RemoveAllRadioButtons

void RemoveAllRadioButtons();

Removes all radio buttons from the task dialog. Reset

void Reset();

Returns the dialog to its default state. UpdateElementText

void UpdateElementText(TASKDIALOG\_ELEMENTS eElement, LPCTSTR pszNewText)

Updates a text element on the Task Dialog. Overridables OnTDButtonClicked

virtual BOOL OnTDButtonClicked(int nButtonID);

Called when the user selects a button or command link. OnTDConstructed

virtual void OnTDConstructed();

Called when the task dialog is constructed, before it is displayed. OnTDCreated

virtual void OnTDCreated();

Called when the task dialog is displayed. OnTDDestroyed

virtual void OnTDDestroyed();

Called when the task dialog is destroyed. OnTDExpandButtonClicked

virtual void OnTDExpandButtonClicked(BOOL bExpanded);

Called when the expand button is clicked. OnTDHelp

virtual void OnTDHelp();

Called when the user presses F1 on the keyboard. OnTDHyperlinkClicked

virtual void OnTDHyperlinkClicked(LPCTSTR pszHref);

Called when the user clicks on a hyperlink. OnTDNavigatePage

virtual void OnTDNavigatePage();

Called when a navigation has occurred. OnTDRadioButtonClicked

virtual BOOL OnTDRadioButtonClicked(int nRadioButtonID);

Called when the user selects a radio button. OnTDTimer

virtual void OnTDTimer(DWORD dwTickCount);

 $Called \ every \ 200 \ milliseconds \ (a proximately) \ when \ the \ TDF\_CALLBACK\_TIMER \ flag \ is \ set. \\ On TDVerification CheckboxClicked$ 

virtual void OnTDVerificationCheckboxClicked(BOOL bChecked);

Called when the user clicks the Task Dialog verification check box. TaskDialogProc

virtual LRESULT TaskDialogProc(UINT uMsg, WPARAM wParam, LPARAM lParam);

Override this function to specify how the messages for this window are to be handled. Return all unhandled messages to TaskDialogProcDefault.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

#### Remarks

Task dialogs require Windows Vista or above. They are not supported on Windows XP. The IsSupported function can be used to test if the operating system is capable of supporting task dialogs.

The following diagram illustrates the various components of a task dialog.

## Summary Information

Header file	taskdialog.h
Win32/64 support	Yes
WinCE support	No

# **CToolTip Class**

## Description

The CToolTip class adds support for the tool tip control. ToolTips appear automatically, or pop up, when the user pauses the mouse pointer over a tool or some other UI element. The ToolTip appears near the pointer and disappears when the user clicks a mouse button, moves the pointer away from the tool, or simply waits for a few seconds.

### **CToolTip Members**



CSize GetBubbleSize(LPT00LINF0 lpToolInfo) const;

Returns the width and height of the ToolTip control. GetDelayTime

int GetDelayTime(DWORD dwDuration) const;

Retrieves the initial, pop-up, and reshow durations currently set for the ToolTip control. GetMargin

void GetMargin(LPRECT lprc) const;

Retrieves the top, left, bottom, and right margins set for the ToolTip window. GetMaxTipWidth

int GetMaxTipWidth() const;

Retrieves the maximum width for the ToolTip window. GetText

void GetText(CString& str, CWnd\* pWnd, UINT\_PTR nIDTool = 0) const;

Retrieves the text from the tool tip control. GetTipBkColor

COLORREF GetTipBkColor() const;

Retrieves the background color in the ToolTip window. GetTipTextColor

COLORREF GetTipTextColor() const;

Retrieves the text color in a ToolTip window. GetToolCount

int GetToolCount() const;

Retrieves a count of the tools maintained by the ToolTip control. GetToolInfo

BOOL GetToolInfo(TOOLINFO& ToolInfo, CWnd\* pWnd, UINT\_PTR nIDTool = 0) c

Retrieves the information that a ToolTip control maintains about the tool. SetDelayTime

void SetDelayTime(UINT nDelay);

void SetDelayTime(DWORD dwDuration, int iTime);

Sets the initial, pop-up, and reshow durations for the ToolTip control  $\operatorname{SetMargin}$ 

void SetMargin(LPRECT lprc);

Sets the top, left, bottom, and right margins for the ToolTip window. SetMaxTipWidth

int SetMaxTipWidth(int iWidth);

Sets the maximum width for the ToolTip window. SetTipBkColor

void SetTipBkColor(COLORREF clr);

Sets the background color in the ToolTip window. SetTipTextColor

void SetTipTextColor(COLORREF clr);

Sets the text color in the ToolTip window. SetToolInfo

void SetToolInfo(LPT00LINF0 lpToolInfo);

Sets the information that the ToolTip control maintains for a tool. Operations Activate

void Activate(BOOL bActivate);

Activates or deactivates the ToolTip control. AddTool

BOOL AddTool(CWnd\* pWnd, UINT nIDText, LPCRECT lpRectTool = NULL, UINT\_P

#### BOOL AddTool(CWnd\* pWnd, LPCTSTR lpszText = LPSTR\_TEXTCALLBACK, LPCRECT

Registers a tool with a ToolTip control. AdjustRect

BOOL AdjustRect(LPRECT lprc, BOOL bLarger = TRUE);

Calculates a ToolTip control's text display rectangle from its window rectangle, or the ToolTip window rectangle needed to display a specified text display rectangle. DelTool

void DelTool(CWnd\* pWnd, UINT\_PTR nIDTool = 0);

Removes a tool from the ToolTip control. HitTest

BOOL HitTest(CWnd\* pWnd, CPoint pt, LPTOOLINFO lpToolInfo) const;

Tests a point to determine whether it is within the bounding rectangle of the specified tool and, if it is, retrieves information about the tool. Pop

void Pop();

Removes a displayed ToolTip window from view. RelayEvent

void RelayEvent(LPMSG lpMsg);

Passes a mouse message to a ToolTip control for processing. SetTitle

B00L SetTitle(UINT uIcon, LPCTSTR lpstrTitle);

Adds a standard icon and title string to a ToolTip. **SetToolRect** 

void SetToolRect(CWnd\* pWnd, UINT\_PTR nIDTool, LPCRECT lpRect);

Sets a new bounding rectangle for the tool. SetWindowTheme

void SetWindowTheme(LPCWSTR lpstrTheme);

Sets the visual style of a ToolTip control. Update

void Update();

Forces the current tool to be redrawn. UpdateTipText

void UpdateTipText(LPCTSTR lpszText, CWnd\* pWnd, UINT\_PTR nIDTool = 0);

void UpdateTipText(UINT nIDText, CWnd\* pWnd, UINT\_PTR nIDTool = 0);

Sets the ToolTip text for a tool.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

### Remarks

ToolTip controls can display a single line of text or multiple lines. Their corners can be rounded or square. They might or might not have a stem that points to the tools like a cartoon speech balloon. ToolTip text can be stationary or can move with the mouse pointer, called tracking. Stationary text can be displayed adjacent to a tool or it can be displayed over a tool, which is referred to as in-place. Standard ToolTips are stationary, display a single line of text, have square corners, and have no stem pointing to the tool.

### **Summary Information**

Header file	controls.h
Win32/64 support	Yes
WinCE support	Yes

## **CTreeView Class**

## Description

A tree-view control is a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional bitmapped image, and each item can have a list of subitems associated with it. By clicking an item, the user can expand or collapse the associated list of subitems.

CTreeView is the class which creates a tree-view control. It can be used as a control in a Dialog, or as the View window in a Frame, Docker, MDI child.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of tree view controls.

#### **CTreeView Members**

Constructor



#### Attributes

GetBkColor	COLORREF GetBkColor() const;
	Retrieves the current background color of the control.
GetChild	HTREEITEM GetChild(HTREEITEM hItem) const;
	Retrieves the first child item of the specified tree-view item.
GetCount	UINT GetCount() const;
	Retrieves a count of the items in a tree-view control.
GetDropHiLightItem	HTREEITEM GetDropHiLightItem() const;
	Retrieves the tree-view item that is the target of a drag-and-drop operation.

GetEditControl	<pre>CEdit* GetEditControl() const;</pre>
	Retrieves the handle to the edit control being used to edit a tree-view item's text.
GetFirstVisible	HTREEITEM GetFirstVisible() const;
	Retrieves the first visible item in a tree-view control window.
GetImageList	<pre>CImageList* GetImageList(int iImageType) const;</pre>
SetimageList	Retrieves the normal or state image list associated with a tree-view control.
GetIndent	UINT GetIndent() const;
Getinuent	Retrieves the amount, in pixels, that child items are indented relative to their parent ite
GetInsertMarkColor	COLORREF GetInsertMarkColor() const;
	Retrieves the color used to draw the insertion mark for the tree-view.
GetItem	BOOL GetItem(TVITEM& Item) const;
	Retrieves some or all of a tree-view item's attributes.
GetItemData	<pre>DWORD_PTR GetItemData(HTREEITEM hItem) const;</pre>
Schembau	Retrieves the application data from a tree-view item.
GetItemHeight	<pre>int GetItemHeight() const;</pre>
Genteimieigin	Retrieves the current height of the tree-view items.
GetItemImage	BOOL GetItemImage(HTREEITEM hItem, int& nImage, int& nSele
Centenninge	Retrieves the index of the normal image and the selected image.
GetItemRect	BOOL GetItemRect(HTREEITEM hItem, CRect& rc, BOOL bTextOn
	Retrieves the bounding rectangle for a tree-view item and indicates whether the item is
GetItemText	CString GetItemText(HTREEITEM hItem, UINT nTextMax /* = 26
	Retrieves the text from a tree-view item.
GetLastVisible	HTREEITEM GetLastVisible() const;
	Retrieves the last expanded item in a tree-view control. This does not retrieve the last i window.
GetNextItem	HTREEITEM GetNextItem(HTREEITEM hItem, UINT nCode) const;
	Retrieves the tree-view item that bears the specified relationship to a specified item.

GetNextSibling	<pre>HTREEITEM GetNextSibling(HTREEITEM hItem) const;</pre>	
	Retrieves the next sibling item of a specified item in a tree-view control.	
GetNextVisible	<pre>HTREEITEM GetNextVisible(HTREEITEM hItem) const;</pre>	
	Retrieves the next visible item that follows a specified item in a tree-view control.	
GetParentItem	<pre>HTREEITEM GetParentItem(HTREEITEM hItem) const;</pre>	
	Retrieves the parent item of the specified tree-view item.	
GetPrevSibling	<pre>HTREEITEM GetPrevSibling(HTREEITEM hItem) const;</pre>	
	Retrieves the previous sibling item of a specified item in a tree-view control.	
GetPrevVisible	<pre>HTREEITEM GetPrevVisible(HTREEITEM hItem) const;</pre>	
	Retrieves the first visible item that precedes a specified item in a tree-view control.	
GetRootItem	HTREEITEM GetRootItem() const;	
	Retrieves the topmost or very first item of the tree-view control.	
GetScrollTime	<pre>int GetScrollTime() const;</pre>	
	Retrieves the maximum scroll time for the tree-view control.	
GetSelection	HTREEITEM GetSelection() const;	
	Retrieves the currently selected item in a tree-view control.	
GetTextColor	COLORREF GetTextColor() const;	
	Retrieves the current text color of the control.	
GetToolTips	CToolTip* GetToolTips() const;	
ottioninpo	Retrieves the handle to the child ToolTip control used by a tree-view control.	
GetVisibleCount	UINT GetVisibleCount() const;	
	Obtains the number of items that can be fully visible in the client window of a tree-	
ItemHasChildren	BOOL ItemHasChildren(HTREEITEM hItem) const;	
	Returns TRUE if the item has children.	
SetBkColor	COLORREF SetBkColor(COLORREF clrBk) const;	
	Sets the background color of the control.	

SetImageList	BOOL SetItemImage(HTREEITEM hItem, int nImage, int nSelec <sup>.</sup>	
	Sets the normal or state image list for a tree-view control and redraws the control using	
SetIndent	<pre>void SetIndent(int indent) const;</pre>	
	Sets the width of indentation for a tree-view control and redraws the control to reflect	
SetInsertMark	BOOL SetInsertMark(HTREEITEM hItem, BOOL fAfter = TRUE) co	
	Sets the insertion mark in a tree-view control.	
SetInsertMarkColor	COLORREF SetInsertMarkColor(COLORREF clrInsertMark) const	
	Sets the color used to draw the insertion mark for the tree view.	
	BOOL SetItem(TVITEM& Item) const;	
SetItem	BOOL SetItem(HTREEITEM hItem, UINT nMask, LPCTSTR szText, int nSelectedImage, UINT nState, UINT nStateMask,	
	Sets some or all of a tree-view item's attributes.	
SetItemData	BOOL SetItemData(HTREEITEM hItem, DWORD_PTR dwData) const	
	Sets the application data for a tree-view item.	
SetItemHeight	<pre>int SetItemHeight(SHORT cyItem) const;</pre>	
	Sets the height of the tree-view items.	
SetItemImage	BOOL SetItemImage(HTREEITEM hItem, int nImage, int nSelec <sup>.</sup>	
	Sets the index of the normal image and the selected image.	
SetItemText	BOOL SetItemText(HTREEITEM hItem, LPCTSTR szText) const;	
JULIUITEAL	Sets the text of a tree-view item.	
SetScrollTime	<pre>UINT SetScrollTime(UINT uScrollTime) const;</pre>	
	Sets the maximum scroll time for the tree-view control.	
SetTextColor	COLORREF SetTextColor(COLORREF clrText) const;	
	Sets the text color of the control.	
SetToolTips	<pre>CToolTip* SetToolTips(CToolTip* pToolTip) const;</pre>	
Section Tips	Sets a tree-view control's child ToolTip control.	

## Operations

	<pre>CImageList* CreateDragImage(HTREEITEM hItem) const;</pre>
CreateDragImage	Creates a dragging bitmap for the specified item in a tree-view control. An application can display the image when dragging the item by using the image list functions.
DeleteAllItems	BOOL DeleteAllItems() const;
	Deletes all items from a tree-view control.
DeleteItem	BOOL DeleteItem(HTREEITEM hItem) const;
	Removes an item and all its children from a tree-view control.
	HWND EditLabel(HTREEITEM hItem) const;
EditLabel	Begins in-place editing of the specified item's text, replacing the text of the item with a single-line edit control containing the text. This function implicitly selects and focuses the specified item.
EndEditLabelNow	B00L EndEditLabelNow(B00L fCancel) const;
	Ends the editing of a tree-view item's label.
EnsureVisible	BOOL EnsureVisible(HTREEITEM hItem) const;
	Ensures that a tree-view item is visible, expanding the parent item or scrolling the tree-view control, if necessary.
	BOOL Expand(HTREEITEM hItem, UINT nCode) const;
Expand	The TreeView_Expand macro expands or collapses the list of child items associated with the specified parent item, if any.
HitTest	HTREEITEM HitTest(TVHITTESTINF0& ht) const;
	Determines the location of the specified point relative to the client area of a tree-view control.
InsertItem	HTREEITEM InsertItem(TVINSERTSTRUCT& tvIS) const;
	Inserts a new item in a tree-view control.
Select	BOOL Select(HTREEITEM hitem, UINT flag) const;
	Selects the specified tree-view item, scrolls the item into view, or redraws the item in the style used to indicate the target of a drag-and-drop operation.
	B00L SelectDropTarget(HTREEITEM hItem) const;
SelectDropTarget	Redraws a specified tree-view control item in the style used to indicate the target of a drag-and-drop operation.

SelectItem	BOOL SelectItem(HTREEITEM hItem) const; Selects the specified tree-view item.	
SelectSetFirstVisible	BOOL SelectSetFirstVisible(HTREEITEM hItem) const; Scrolls the tree-view control vertically to ensure that the specified item is visible. If possible, the specified item becomes the first visible item at the top of the control's	
SortChildren	window. BOOL SortChildren(HTREEITEM hItem, BOOL fRecurse) const; Sorts the child items of the specified parent item in a tree-view control.	
SortChildrenCB	BOOL SortChildrenCB(TVSORTCB& sort, BOOL fRecurse) const; Sorts tree-view items using an application-defined callback function that compares the items.	

#### **Overridables**

	<pre>virtual void PreRegisterClass(WNDCLASS &amp;wc);</pre>
	Set the window class parameters before the window is created.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

#### Remarks

Like all common controls, the Tree-VIew control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a Tree-View control.

Refer to the explorer sample to see a demonstration of the CListView and CTreeView classes.

### **Summary Information**

Header file	treeview.h	
Win32/64	Yes	
support	100	
WinCE support	Yes	

Library required Comctl32.lib

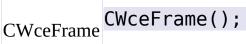
## **CWceFrame Class**

## Description

The WceFrame class provides a simple frame for the Windows CE operating system. It uses a command bar to provide the menu and toolbar buttons.

Use the PocketPCFrame sample as the starting point for your own frame based application for Windows CE.

#### **CWceFrame Members**



Constructor for CWceFrame.

#### AddToolBarButton

virtual void AddToolBarButton(UINT nID);

Adds a button to the toolbar. GetViewRect

CRect GetViewRect() const;

Returns a RECT structure which contains the dimensions of the client area of the frame.

#### GetMenuBar

CCmdBar& GetMenuBar() const;

Returns a reference to the frame's CCmdBar object, which manages the frame's command bar. RecalcLayout

virtual void RecalcLayout();

Repositions the client area of the frame. SetButtons

virtual void SetButtons(const std::vector<UINT> ToolBarData);

Sets the toolbar buttons of the command bar.

#### **Overridables**

OnActivate	<pre>virtual void OnActivate(WPARAM wParam, LPARAM lParam); Called when the frame is activated.</pre>	
OnCreate	<pre>virtual void OnCreate(); Called during window creation. Override this function to perform tasks such as creating child windows.</pre>	
PreCreate	<pre>virtual void PreCreate(CREATESTRUCT &amp;cs); Called before the window is created. Override this function to set the window creation parameters.</pre>	
PreTranslateMessage	<pre>virtual BOOL PreTranslateMessage(MSG* pMsg);</pre>	

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## **Summary Information**

Header file	wceframe.h
Win32/64 support	No
WinCE support	Yes

## **CWebBrowser Class**

## Description

This class provides a web browser in a window. It can be used to provide a web browser in any place a view window is used.

#### **CWebBrowser Members**

Construction

CWebBrowser

CWebBrowser();

Constructs a CWebBrowser object. Attributes GetApplication

```
LPDISPATCH GetApplication() const;
```

Retrieves the automation object for the application that is hosting the WebBrowser Control.

#### GetAXWindow

CAXWindow& GetAXWindow() const;

Returns a reference to the ActiveX container window. GetBusy

B00L GetBusy() const;

Retrieves a value that indicates whether the object is engaged in a navigation or downloading operation. GetContainer

LPDISPATCH GetContainer() const;

Retrieves an object reference to a container. GetFullScreen

B00L GetFullScreen() const;

Retrieves a value that indicates whether Internet Explorer is in full-screen mode or normal window mode. GetHeight

long GetHeight() const;

Retrieves the height of the object. GetIWebBrowser2

IWebBrowser2\* GetIWebBrowser2();

Returns the pointer to the IWebBrowser interface. GetLeft

long GetLeft() const;

Retrieves the coordinate of the left edge of the object. GetLocateName

CString GetLocationName() const;

Retrieves the path or title of the resource that is currently displayed. GetLocationURL

CString GetLocationURL() const;

Retrieves the URL of the resource that is currently displayed. GetOffline

B00L GetOffline() const;

Retrieves a value that indicates whether the object is operating in offline mode. GetReadyState

READYSTATE GetReadyState() const;

Retrieves the ready state of the object. GetRegisterAsBrowser

B00L GetRegisterAsBrowser() const;

Retrieves a value that indicates whether the object is registered as a top-level browser window. **GetTheaterMode** 

B00L GetTheaterMode() const;

Retrieves the theater mode state of the object. GetTop

long GetTop() const;

Retrieves the coordinate of the top edge of the object. GetTopLevelContainer

B00L GetTopLevelContainer() const;

Retrieves a value that indicates whether the object is a top-level container. GetType

CString GetType() const;

Retrieves the user type name of the contained document object. GetVisible

B00L GetVisible() const;

Retrieves a value that indicates whether the object is visible or hidden. GetWidth

long GetWidth() const;

Retrieves the width of the object. SetFullScreen

void SetFullScreen(BOOL bNewValue);

Sets a value that indicates whether Internet Explorer is in full-screen mode or normal window mode. SetHeight

void SetHeight(long nNewValue);

Sets the height of the object. SetLeft

void SetLeft(long nNewValue);

Sets the coordinate of the left edge of the object.e. SetOffline

void SetOffline(BOOL bNewValue);

Sets a value that indicates whether the object is operating in offline mode.

#### SetRegisterAsBrowser

void SetRegisterAsBrowser(BOOL bNewValue);

Sets a value that indicates whether the object is registered as a top-level browser window.

#### SetTheaterMode

void SetTheaterMode(BOOL bNewValue);

Sets the theatre mode state of the object. SetTop

void SetTop(long nNewValue);

Sets the coordinate of the top edge of the object. SetVisible

void SetVisible(BOOL bNewValue);

Sets a value that indicates whether the object is visible or hidden. SetWidth

void SetWidth(long nNewValue);

Sets the width of the object. **Operations** 

AddWebBrowserControl

void AddWebBrowserControl(void);

Adds the IWebBrowser interface to the ActiveX container window. Exec

void ExecWB(OLECMDID cmdID, OLECMDEXECOPT cmdexecopt, VARIANT\* pvaIn, VA

Executes a command using the IOleCommandTarget interface. GetProperty

BOOL GetProperty(LPCTSTR pszProperty, CString& strValue);

VARIANT GetProperty( LPCTSTR pszProperty);

Gets the value associated with the specified property name. GoBack

void GoBack();

Navigates backward one item in the history list. GoForward

void GoForward();

Navigates forward one item in the history list. GoHome

void GoHome();

Navigates to the current home or start page. GoSearch

void GoSearch();

Navigates to the current search page. Navigate

void Navigate(LPCTSTR pszURL, DWORD dwFlags = 0, LPCTSTR pszTargetFrameN LPCTSTR pszHeaders = NULL, LPVOID pvPostData = NULL, DWORD dwPostDa

Navigates to a resource identified by a URL or to a file identified by a full path. Navigate2

void Navigate2(LPITEMIDLIST pIDL, DWORD dwFlags = 0, LPCTSTR pszTargetFr

void Navigate2(LPCTSTR pszURL, DWORD dwFlags = 0, LPCTSTR pszTargetFrame LPCTSTR pszHeaders = NULL, LPVOID lpvPostData = NULL, DWORD dwPostD

Navigates the browser to a location specified by a pointer to an item identifier list (PIDL) for an entity in the Microsoft Windows Shell namespace, or a URL. **PutProperty** 

void PutProperty(LPCTSTR pszPropertyName, const VARIANT& vtValue);

void PutProperty(LPCTSTR pszPropertyName, double dValue);

void PutProperty(LPCTSTR pszPropertyName, long lValue);

void PutProperty(LPCTSTR pszPropertyName, LPCTSTR lpszValue);

void PutProperty(LPCTSTR pszPropertyName, short nValue);

Sets the value of a property associated with the object. Refresh

void Refresh();

Reloads the file that is currently displayed in the object. Refresh2

void Refresh2(int nLevel);

Reloads the file that is currently displayed with the specified refresh level. Stop

void Stop();

Cancels a pending navigation or download, and stops dynamic page elements, such as background sounds and animations. **Overridables** OnCreate

```
virtual void OnCreate();
```

Called when the window is created. OnSize

virtual void OnSize(int width, int height);

Called when the window is resized.

#### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

## Remarks

Refer to the Browser sample for a demonstration of how to use CWebBrowser to implement a simple web browser.

Header file	webbrowser.h	
Win32/64	Yes	
support	103	

WinCE support	NO
Library required	

# **CWinApp Class**

# Description

The class responsible for initializing Win32++. You inherit from this class to start the application.

# **CWinApp Members**

#### Construction

CWinApp	CWinApp();
	Constructs a CWinApp object.

## Attributes

GetInstanceHandle	HINSTANCE GetInstanceHandle() const; Returns the instance handle (HINSTANCE) of the application
GetResourceHandle Returns the instance handle of resources. This can the the HINSTANCE of the application or the HINSTANCE of a resource dll.	
SetResourcertailule	<pre>void SetResourceHandle(HINSTANCE hResource); Sets the instance handle of resources.</pre>

## Operations

I IO	LoadCursor(LPCTSTR lpszResourceName) const;
LoadCursor	LoadCursor(int nIDCursor) const;
	Loads the specified cursor. The cursor is defined in the resource script (resource.rc).
	<pre>HCURSOR LoadStandardCursor(LPCTSTR lpszCursorName) const;</pre>

LoadStandardCursor Returns the handle of a standard cursor. Standard cursors include: IDC_APPSTARTING, IDC_ARROW, IDC_CROSS, IDC_HAND, IDC_HELP, IDC_IBEAM, IDC_NO, IDC_SIZEALL, IDC_SIZENESW, IDC_SIZENS, IDC_SIZENWSE, IDC_SIZEWE, IDC_UPARROW, IDC_WAIT.	
HICON LoadIcon(LPCTSTR lpszResourceName) const; LoadIcon HICON LoadIcon(int nIDIcon) const; Loads the icon resource whose size conforms to the SM_CXICON and SM_CYICON system metric values. For other icon sizes, use the LoadImage windows API function.	
LoadImage HA Loa and	NDLE LoadImage(LPCTSTR lpszResourceName, UINT uType, int cx, i NDLE LoadImage(int nIDImage, UINT uType, int cx, int cy, UINT ads an icon, cursor, animated cursor or bitmap image. uType can be IMAGE_BITMAP, IMA( height in pixels. fuLoad can be LR_DEFAULTCOLOR, LR_CREATEDIBSECTION, LR_I _LOADMAP3DCOLORS, R_LOADTRANSPARENT, LR_MONOCHROME, LR_SHARE
LoadStandardIcon HICON LoadStandardIcon(LPCTSTR lpszIconName) const; Returns the handle of a standard Icon. Standard Icons include: IDI_APPLICATION, IDI_ASTERISK, IDI_ERROR, IDI_EXCLAMATION, IDI_HAND, IDI_INFORMATION, IDI_QUESTION, IDI_WARNING	
SetCursor	HCURSOR SetCursor(HCURSOR hCursor) const; Sets the current cursor and returns the previous one. Note: The cursor will be set to the window's class cursor (if one is set) each time the mouse is moved over the window. You can specify different cursors for different conditions while processing WM_SETCURSOR.

## **Overridables**

InitInstance	<pre>virtual BOOL InitInstance();</pre>
	This function is called when the application starts. Override this to perform tasks such as creating a window.
Run	virtual int Run();
	Calls InitInstance and runs the message loop. Use this in WinMain to run the application.

#### **Base class Members**

•

For base class members, refer to the members of <u>CWinThread.htm</u>

## Remarks

## Starting a Win32++ application

CWinApp (or a class inherited from CWinApp) must be used to run a Win32++ application. Here we see a simple example of a class inherited from CWinApp.

```
// The class inherited from CWinApp which starts Win32++
class CSimpleApp : public CWinApp
{
public:
 CSimpleApp() {}
 virtual ~CSimpleApp() {}
 virtual BOOL InitInstance();
private:
 CView m_View;
};
BOOL CSimpleApp::InitInstance()
ł
 // This function is called by Run
 m View.Create(); // Create the Window
 return TRUE;
}
```

Notice that InitInstance is used to specify what happens when we use the Run function. In this case it is used to create the CView window.

## Running a Win32++ application

Once we have our CWinApp derived class, we use it in WinMain start our application. The following code demonstrates a typical use of the class inherited from CWinApp.

```
// Run the application until the window is destroyed
return MyApp.Run();
```

Note that this simple program has two key steps

- Constructs MyApp, which is a CSimpleApp object. CSimpleApp is inherited from WinApp.
- Uses the Run function to call MyApp's InitInstance, and run the message loop.

Separating the construction and the running of CWinApp like this allows the CWinApp derived class (including all member objects) to be fully constructed, before attempting to do things which could generate exceptions, such as creating windows.

### A complete simple application

}

Sometimes it is easier to fit the pieces together when we see a complete application rather than a collection of code snippets. The following code sample is a complete simple windows application. It creates an ordinary window, and ends the application when the window is destroyed.

```
#include "../Win32++/Wincore.h"
// A class inherited from CWnd for the window
class CView : public CWnd
{
public:
 CView() {}
 virtual ~CView() {}
 virtual LRESULT WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lP
};
LRESULT CView::WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lPara
Ł
 // Handle the messages for this window
 switch (uMsg)
 {
 case WM DESTROY:
   //End the program when window is destroyed
   ::PostQuitMessage(0);
   break;
```

```
}
 //Use the default message handling for remaining messages
 return WndProcDefault(hWnd, uMsg, wParam, lParam);
}
// The class inherited from CWinApp which starts Win32++
class CSimpleApp : public CWinApp
{
public:
 CSimpleApp() {}
 virtual ~CSimpleApp() {}
 virtual BOOL InitInstance();
private:
 CView m View;
};
BOOL CSimpleApp::InitInstance()
{
 // This function is called by Run
 m View.Create(); // Create the Window
 return TRUE;
}
// The entry point for our windows application
INT WINAPI WinMain(HINSTANCE, HINSTANCE, LPTSTR, int)
{
 // Start Win32++
 CSimpleApp MyApp;
 // Run the application until the window is destroyed
 return MyApp.Run();
}
```

Header file wincore.h	
Win32/64	Yes
support	105
WinCE support	Yes

# **CWindowDC** Class

## Description

The class responsible for creating a device context for the entire area of a window, including the non-client area.

## **CWindowDC Members**

#### Initialization and Assignment

Initialization and Assignment	
	CWindowDC(const CWnd* pWnd);
	Constructs a CWindowDC object.

#### **Base class Members**

For base class members, refer to the members of <u>CDC</u>.

## Remarks

Refer to the <u>Graphics Device Interface</u> section of Using Win32++ for additional information on using this class.

Header file	gdi.h
Win32/64	Yes
support	
WinCE support	Yes

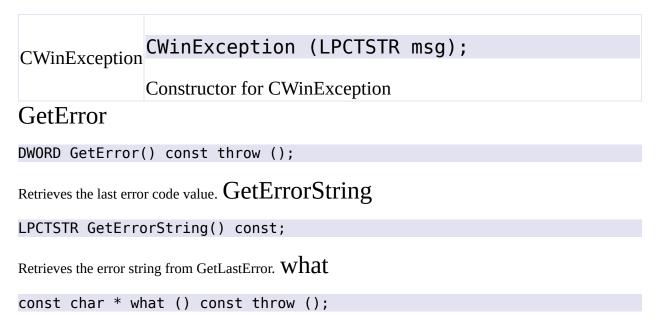
# **CWinException Class**

# Description

A class which handles exceptions. Exceptions should only be used to handle errors which might arise from exceptional circumstances. In Win32++, exceptions are typically used to handle the unexpected failure of Windows API functions, like, for example, CreateWindowEx.

It is not normal for a program to routinely throw exceptions. A thrown exception often indicates a programming error, but they could also be thrown when a system is so starved of resources that the operating system cannot successfully allocate the resources needed to complete the task.

# **CWinException Members**



Sends the exception message and error string to the debug window.

## Remarks

Exceptions should be caught by constant reference to avoid the creation of a temporary copy of the CWinException object. Since an exception could be thrown when the operating system is already starved of resources, consuming additional resources to create this copy to catch the exception would by very undesirable.

```
void SomeFunction()
{
 try
  {
   if(IsValid())
    {
      // Do the normal stuff
    }
   else
      throw CWinException(_T("Not Valid"));
 }
 catch (const CWinException &e)
  {
   // Send the exception information to the debug window
   e.what();
  }
}
```

The what function reports the text message associated with the exception and sends this information to the IDE's debug window. It also reports the text associated with the GetLastError function.

Header file wincore.h	
Win32/64 support	Yes
WinCE support	Yes

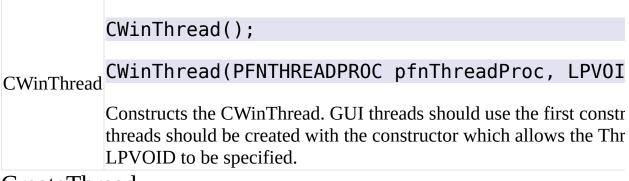
# **CWinThread Class**

# Description

A program or process can contain one or more threads. Threads run independantly of each other, and can run concurrently. When used wisely, the use of multiple threads can make better use of the computer's resources, particularly multiple CPUs, allowing an application to be more responsive. The CWinThread class simplifies the use of threads with Win32++.

## **CWinThread Members**

#### Operations



## CreateThread

```
void CreateThread(unsigned initflag = 0, unsigned stack_size = 0, LPSECU
```

Creates the thread. Valid argument values: initflag - Either CREATE\_SUSPENDED or 0 stack\_size - Either the stack size or 0

 $pSecurityAttributes - Either a pointer to SECURITY\_ATTRIBUTES or 0 \\ GetAccelerators$ 

HACCEL GetAccelerators() const;

Returns the handle to the accelerator table. GetAcceleratorsWindow

CWnd\* GetAcceleratorsWindow() const;

Returns a pointer to the window for accelerator keys. GetThread

HANDLE GetThread() const;

Returns the thread handle. GetThreadID

int GetThreadID() const;

Returns the thread ID. GetThreadPriority

int GetThreadPriority() const;

Returns the thread's priority. ResumeThread

DWORD ResumeThread() const;

Resumes the execution of the thread. By default, the thread is created in a suspended state, and should be resumed when the application is ready to have the thread run. SetAccelerators

void SetAccelerators(HACCEL hAccel, CWnd\* pWndAccel) const;

Sets handle of the accelerator table, and a pointer to the window used for accelerator keys. **SetThreadPriority** 

BOOL SetThreadPriority(int nPriority) const;

Sets the thread's priority. SuspendThread

DWORD SuspendThread() const;

Suspends the execution of the thread.

### **Overridables**

InitInstance	<pre>virtual BOOL InitInstance(); The function which runs when a GUI thread starts. Override this function to specify</pre>
	what the thread does when it starts. Return TRUE if the message loop should run, otherwise return FALSE.
	<pre>virtual int MessageLoop();</pre>
MessageLoop	The MessageLoop function is called if InitInstance returns TRUE. Override this function if you wish to customise the message loop run by this thread.
	<pre>virtual BOOL OnIdle(LONG lCount);</pre>
OnIdle	OnIdle is called to perform idle processing. The lCount is incremented each time OnIdle is called while the message queue is empty. Override OnIdle to perform small tasks. Larger tasks can be split up into pieces which are processed when lCount

	reaches specific values. Return TRUE to perform additional idle tasks, and return FALSE when there are no more tasks to perform.
PreTranslateMessage	<pre>virtual BOOL PreTranslateMessage(MSG Msg); PreTranslateMessage processes the keyboard accelerator keys and calls CWnd::PreTranslateMessage for keyboard and mouse events.</pre>

## Remarks

CWinThread supports the creation of both worker threads and GUI (Graphic User Interface) threads.

## **GUI Threads:**

GUI threads (i.e. threads that have windows) should inherit from this class, use the first constuctor, and override InitInstance to specify what the thread does when it starts. If your thread is used to run one or more windows, InitInstance should return TRUE to start the MessageLoop.

### Worker Threads:

Worker threads don't have windows and don't need a message loop. Worker threads don't need to inherit from this class. They can simply use the constructor which allows the ThreadProc and LPVOID to be specified.

### Note:

Creating an instance of CWinThread doesn't create the thread. The CreateThread member function is used to create the thread.

It is your job to end the thread before CWinThread ends! To end a thread with a message loop, post a WM\_QUIT message to the thread. To end a thread without a message loop, set an event, and end the thread when the event is received.

#### **Programming Hints:**

- It is never a good idea to use things like TerminateThread or ExitThread to end your thread. These represent poor programming techniques, and are likely to leak memory and resources.
- Avoid using SendMessage between threads, as this will cause one thread to wait for the other to respond. Use PostMessage between threads to avoid this problem.

- Access to variables and resources shared between threads need to be made thread safe. Having one thread modify a resource or variable while another thread is accessing it is a recipe for disaster.
- Thread Local Storage (TLS) can be used to replace global variables to make them thread safe. With TLS, each thread gets its own copy of the variable.
- Critical Sections can be used to make shared resources thread safe.
- Window messages (including user defined messages) can be posted between GUI threads to communicate information between them.
- Events (created by CreateEvent) can be used to communicate information between threads (both GUI and worker threads).
- Avoid using sleep to synchronise threads. The various wait functions (e.g. WaitForSingleObject) will be better for thread synchronisation.

#### **About Threads:**

Each program that executes has a "process" allocated to it. A process has one or more threads. Threads run independently of each other. It is the job of the operating system to manage the running of the threads, and do the task switching between threads as required. Systems with multiple CPUs will be able to run as many threads simultaneously as there are CPUs.

Threads behave like a program within a program. When the main thread starts, the application runs the WinMain function and ends when WinMain ends. When another thread starts, it too will run the function provided to it, and end when that function ends.

Header file	wincore.h
Win32/64 support	Yes
WinCE support	Yes

# **CWnd Class**

# Description

The CWnd class represents a window. CWnd handles such things as window creation and window destruction, as well as determining how the window messages are handled. Each window created by Win32++ is a CWnd object, or an object derived from CWnd. Many of the classes in Win32++ are inherited from CWnd, including <u>CFrame</u>, <u>CMDIChild</u>, <u>CMDIFrame</u>, <u>CMenuBar</u>, <u>CReBar</u>, <u>CStatusBar</u>, and <u>CToolBar</u>.

Inherit from this class in your own application to create windows. The WndProc member function is called whenever the window associated with the CWnd object receives a Window message. Override WndProc to handle window messages, and pass all unhandled messages on to WndProcDefault.

## **CWnd Members:**

#### Construction

CWnd	CWnd();
	Constructs a CWnd object

### **Operations**

operator HWND	operator HWND() const;
	Returns the HWND belonging to this CWnd.
Attach	<pre>virtual BOOL Attach(HWND hWnd);</pre>
	Attaches a CWnd object to an existing window.
AttachDlgItem	<pre>virtual BOOL AttachDlgItem(UINT nID, CWnd* pParent); Attaches a CWnd object to a dialog item.</pre>
	<pre>virtual void CenterWindow() const;</pre>
CenterWindow	Positions the window over the center of its parent.

Create	virtual HWND Create(CWnd* pParent = NULL);
	Creates the window with default parameters. Use PreCreate to set the default parame
	virtual HWND CreateEx(DWORD dwExStyle, LPCTSTR lpszClass LPCTSTR lpszWindowName, DWORD dwSt int x, int y, int nWidth, int nHei HWND hwndParent, HMENU nIDorHMenu,
CreateEx	virtual HWND CreateEx(DWORD dwExStyle, LPCTSTR lpszClass LPCTSTR lpszWindowName, DWORD dwSty CWnd* pParent, UINT nID, LPVOID lpP
	Creates the window by specifying each parameter.
Destroy	<pre>virtual void Destroy();</pre>
	Destroys the window associated with the CWnd, and returns the CWnd to its default
Detach	virtual HWND Detach();
	Detaches a window from a CWnd object. It reverses an attach.
	<pre>static CWnd* FromHandle(HWND hWnd);</pre>
FromHandle	Returns the CWnd associated with the specified window handle. If a CWnd object d a temporary CWnd object is created. This temporary CWnd will be deleted sometim message is complete.
	<pre>static CWnd* FromHandlePermanent(HWND hWnd);</pre>
FromHandlePermanent	Returns the CWnd associated with the specified window handle. If a CWnd object d this function returns NULL.
	virtual LRESULT FinalWindowProc(UINT uMsg, WPARAM wParam
FinalWindowProc	Passes messages on to the appropriate default window procedure. When a message i either return FinalWindowProc or a specific value recommended in the Windows AI
Invalidate	<pre>virtual void Invalidate(BOOL bErase = TRUE) const;</pre>
	Invalidates the entire window. The window will be redrawn when the next WM_PAI
SetIconLarge	<pre>virtual HICON SetIconLarge(int nIcon);</pre>
	Loads the large icon from the specified resource, and assigns it to the window.
SetIconSmall	<pre>virtual HICON SetIconSmall(int nIcon);</pre>
	Loads the small icon from the specified resource, and assigns it to to the window.
WndProcDefault	virtual LRESULT WndProcDefault(UINT uMsg, WPARAM wParam,

## Overridables

	<pre>virtual void OnAttach();</pre>
OnAttach	Called when a window is attached to the CWnd. Note that window controls are subclacreated, and call OnAttach.
OnCommand	<pre>virtual BOOL OnCommand(WPARAM wParam, LPARAM lParam);</pre>
Oncommand	Called when the user selects a command, typically in response to a menu or toolbar bu
	<pre>virtual void OnClose();</pre>
OnClose	Called in response to WM_CLOSE. Override to suppress destroying the window. A W SendMessage(WM_SYSCOMMAND, SC_CLOSE, 0) or by clicking the X to close the
OnCreate	<pre>virtual void OnCreate();</pre>
OnCreate	Called during window creation. Override this function to perform tasks such as creatin that window controls are attached after they are created, and don't call OnCreate.
	<pre>virtual void OnDestroy();</pre>
OnDestroy	This function is called when a window is destroyed. Override it to do additional tasks, application with PostQuitMessage
	<pre>virtual void OnDraw(CDC* pDC);</pre>
OnDraw	Called when part of the window is about to be redrawn. Override this function to speci should be drawn.
	<pre>virtual BOOL OnEraseBkgnd(CDC* pDC);</pre>
OnEraseBkGnd	Called when part of the window background is erased. Override this function to specif background should be drawn.
OnInitialUpdate	<pre>virtual void OnInitialUpdate();</pre>
1	Called after the window is created. Override this function to perform tasks once the wi
	<pre>virtual void OnMenuUpdate(UINT nID);</pre>
OnMenuUpdate	Called when a popup menu is about to be displayed. Override this function to set or cleradio button to menu items.
	virtual LRESULT OnMessageReflect(UINT uMsg, WPARAM wParam
OnMessageReflect	Called when a notification message is sent to the parent. Override this function to han

	messages in the CWnd that generated them.
OnNotify	virtual LRESULT OnNotify(WPARAM wParam, LPARAM lParam); Called when a WM_NOTIFY message is received from a child window. Override this notifications in the parent's CWnd.
OnNotifyReflect	virtual LRESULT OnNotifyReflect(WPARAM wParam, LPARAM lPa Called when a WM_NOTIFY message is sent to the parent. Override this function to f the CWnd that generated them.
Precreate	<pre>virtual void PreCreate(CREATESTRUCT&amp; cs); Override this function to set the window creation parameters used by Create.</pre>
PreregisterClass	virtual void PreRegisterClass(WNDCLASS& wc); Override the function to set the window class parameters used by Create.
PreTranslateMessage	<pre>virtual BOOL PreTranslateMessage(MSG* pMsg); Override this function to filter mouse and keyboard messages prior to being passed to</pre>
WndProc	virtual LRESULT WndProc(UINT uMsg, WPARAM wParam, LPARAM Override this function to specify how the messages for this window are to be handled. messages to WndProcDefault.

## Attributes

FromHandle	<pre>static CWnd* FromHandle(HWND hWnd);</pre>
	Returns a pointer to the CWnd object, given the window handle (HWND).
GetHwnd	HWND GetHwnd() const;
	Returns the handle of the window associated with CWnd.
GetPrevWindowProc	WNDPROC GetPrevWindowProc() const;
Getriev windowrioc	Pointer to the Window Procedure prior to it being subclassed.

## **Member Variables**

m\_hWnd

HWND m\_hWnd;

## Wrappers for Window API functions and messages (don't override these).

BeginPaint	CDC* BeginPaint(PAINTSTRUCT& ps) const;
	Prepares the specified window for painting and fills a PAINTSTRUCT structure painting.
	BOOL BringWindowToTop() const;
BringWindowToTop	Brings the window to the top of the Z order. If the window is a top-level window a child window, the top-level parent window associated with the child window is
CallWindowProc	LRESULT CallWindowProc(WNDPROC lpPrevWndFunc, UINT Msg WPARAM wParam, LPARAM lParam)
	Passes message information to the specified window procedure.
CheckDlgButton	BOOL CheckDlgButton(int nIDButton, UINT uCheck) const
CheckDigDutton	Changes the check state of a button control.
CheckRadioButton	BOOL CheckRadioButton(int nIDFirstButton, int nIDLastl int nIDCheckButton) const;
CheckradioDutton	Adds a check mark to (checks) a specified radio button in a group and removes a other radio buttons in the group.
	CWnd* ChildWindowFromPoint(POINT pt) const;
ChildWindowFromPoint	Determines which, if any, of the child windows belonging to a parent window consearch is restricted to immediate child windows. Grandchildren, and deeper desc searched.
	B00L ClientToScreen(P0INT& pt) const;
ClientToScreen	BOOL ClientToScreen(RECT& rc) const;
	Converts the client-area coordinates of a specified point to screen coordinates
CloseWindow	BOOL CloseWindow() const;
CIOSE WILLIOW	Minimizes (but does not destroy) the window. This function is not supported on
DefWindowProc	LRESULT DefWindowProc(UINT uMsg, WPARAM wParam, LPARAI Calls the default window procedure to provide default processing for any windo does not process.
	· · ·

	HDWP DeferWindowPos(HDWP hWinPosInfo, const CWnd* pIn int x, int y, int cx, int cy, UIN
DeferWindowPos	HDWP DeferWindowPos(HDWP hWinPosInfo, const CWnd* pIn const RECT& rc, UINT uFlags) const
	Updates the multiple window position structure for the window. The function thupdated structure.
DlgDirList	<pre>int DlgDirList(LPTSTR lpPathSpec, int nIDListBox,</pre>
	Replaces the contents of a list box with the names of the subdirectories and files can filter the list of names by specifying a set of file attributes. This function is r
DlgDirListComboBox	<pre>int DlgDirListComboBox(LPTSTR lpPathSpec, int nIDCombo int nIDStaticPath, UINT uFilety</pre>
	Replaces the contents of a combo box with the names of the subdirectories and 1 can filter the list of names by specifying a set of file attributes. This function is 1
	BOOL DlgDirSelectEx(LPTSTR lpString, int nCount, int
DlgDirSelectEx	Retrieves the current selection from a single-selection list box. It assumes that the DlgDirList function and that the selection is a drive letter, filename, or directory supported on WinCE.
DlgDirSelectComboBoxEx	BOOL DlgDirSelectComboBoxEx(LPTSTR lpString, int nCou int nIDComboBox) const;
	Retrieves the current selection from a combo box filled by using the DlgDirList(selection is interpreted as a drive letter, a file, or a directory name. This function
DrawAnimatedRects	BOOL DrawAnimatedRects(int idAni, RECT& rcFrom, RECT&
	Draws a wire-frame rectangle and animates it to indicate the opening of an icon maximizing of a window. This function is not supported on WinCE.
DrawCaption	BOOL DrawCaption(CDC* pDC, RECT& rc, UINT uFlags) con
	Draws a window caption. This function is not supported on WinCE.
DrawMenuBar	BOOL DrawMenuBar() const;
	Redraws the menu bar of the window. If the menu bar changes after the system l function must be called to draw the changed menu bar.
EnableScrollBar	B00L EnableScrollBar(UINT uSBflags, UINT uArrows) con
	Enables or disables one or both scroll bar arrows. This function is not supported
	BOOL EnableWindow(BOOL bEnable = TRUE) const;

EnableWindow	The EnableWindow function enables or disables mouse and keyboard input When input is disabled, the window does not receive input such as mouse c enabled, the window receives all input.
EndPaint	BOOL EndPaint(PAINTSTRUCT& ps) const;
	Marks the end of painting in the specified window. This function is required function, but only after painting is complete.
GetActiveWindow	CWnd* GetActiveWindow() const;
	Returns the pointer to the active window. This pointer might be temporary,
GetAncestor	$CWnd^*$ GetAncestor(UINT gaFlag = 3 /*= GA_ROOTOWNER;
	Returns the pointer to the ancestor window. This pointer might be temporary
	CWnd* GetCapture() const;
GetCapture	Returns the pointer to the window (if any) that has captured the mouse. This don't save it for later use.
GetClassLongPtr	<pre>ULONG_PTR GetClassLongPtr(int nIndex) const;</pre>
	Retrieves the specified value from the WNDCLASSEX structure associated
GetClassName	LPCTSTR GetClassName() const;
	Returns a pointer to a TCHAR array containing the class name of the windo
	<pre>CRect GetClientRect() const;</pre>
GetClientRect	Retrieves the coordinates of a window's client area. The client coordinates s corners of the client area. Because client coordinates are relative to the upper area, the coordinates of the upper-left corner are $(0,0)$ .
GetDC	CDC* GetDC() const;
GEDC	Retrieves a pointer to the display device context (CDC) for the client area o
	CDC* GetDCEx(HRGN hrgnClip, DWORD flags) const;
GetDCEx	Retrieves a pointer to a display device context (CDC) for the client area of t extension to the GedDC function, which gives an application more control o occurs in the client area.
GetDesktopWindow	CWnd* GetDesktopWindow() const;
	Retrieves a pointer to the desktop window. This pointer might be temporary
GetDlgCtrlID	<pre>int GetDlgCtrlID() const;</pre>

	Retrieves the control ID value for any child window.
GetDlgItem	CWnd* GetDlgItem(int nIDDlgItem) const;
	Retrieves a pointer to a control in the dialog box. This pointer might be tem
GetDlgItemInt	<pre>UINT GetDlgItemInt(int nIDDlgItem, B00L* lpTranslat</pre>
occo igricilia	Translates the text of a specified control in a dialog box into an integer value
GetDlgItemText	LPCTSTR GetDlgItemText(int nIDDlgItem) const;
GetDigitemitext	Retrieves the title or text associated with a control in a dialog.
GetFont	CFont* GetFont() const;
Getront	Retrieves a handle to the font assigned to the window.
GetIcon	HICON GetIcon(BOOL bBigIcon) const;
Gencon	Retrieves a handle to the icon assigned to the window.
	<pre>CWnd* GetLastActivePopup() const;</pre>
GetLastActivePopup	Retrieves the pointer to the pop-up window owned by the window which we might be temporary, so don't save it for later use.
GetMenu	<pre>CMenu* GetMenu() const;</pre>
Octivicitu	Retrieves a handle to the menu assigned to the window. This function is not
	CWnd* GetNextDlgGroupItem(CWnd* pCtl, BOOL bPreviou
GetNextDlgGroupItem	Retrieves a handle to the first control in a group of controls that precedes (o dialog box.
	CWnd* GetNextDlgTabItem(CWnd* pCtl, B00L bPrevious)
GetNextDlgTabItem	Retrieves a handle to the first control that has the WS_TABSTOP style that control.
	CWnd* GetParent() const;
GetParent	Retrieves a pointer to the window's parent or owner (if any). This pointer m for later use.
	BOOL GetScrollInfo(int fnBar, SCROLLINFO& si) const
GetScrollInfo	Retrieves the parameters of a scroll bar, including the minimum and maxim and the position of the scroll box (thumb).
	<pre>int GetScrollPos(int nBar) const;</pre>

GetScrollPos	Retrieves the current position of the scroll box (thumb) in the specified scroll bar relative value that depends on the current scrolling range. For example, if the sc and the scroll box is in the middle of the bar, the current position is 50. This fun WinCE. New applications should use the GetScrollInfo function.	
GetScrollRange	BOOL GetScrollRange(int nBar, int& MinPos, int& MaxPo Retrieves the current minimum and maximum scroll box (thumb) positions for function is not supported on WinCE. New applications should use the GetScro	
GetSystemMenu	CMenu* GetSystemMenu(BOOL bRevert) const; Allows the application to access the window menu (also known as the system r copying and modifying. This function is not supported on WinCE.	
GetTopWindow	CWnd* GetTopWindow() const; Retrieves a pointer to the child window at the top of the Z order. This pointer it for later use.	
GetUpdateRect	CRect GetUpdateRect(BOOL bErase) const; Retrieves the coordinates of the smallest rectangle that completely encloses the window.	
GetUpdateRgn	<pre>int GetUpdateRgn(CRgn* pRgn, BOOL bErase) const; Retrieves the update region of a window by copying it into the specified region.</pre>	
GetWindow	CWnd* GetWindow(UINT uCmd) const; Retrieves a pointer to a window that has the specified relationship (Z-Order of pointer might be temporary, so don't save it for later use.	
GetWindowDC	CDC* GetWindowDC() const; Retrieves the device context (DC) for the entire window, including title bar, redevice context permits painting anywhere in a window, because the origin of corner of the window instead of the client area.	
GetWindowLongPtr	LONG_PTR GetWindowLongPtr(int nIndex) const; Retrieves information about the window. The function also retrieves the value window memory.	
GetWindowPlacement	BOOL GetWindowPlacement(WINDOWPLACEMENT& pWndpl) cons Retrieves the show state and the restored, minimized, and maximized positions not supported on WinCE.	
GetWindowRect	CRect GetWindowRect() const;	

	Retrieves the dimensions of the bounding rectangle of the specified window. coordinates that are relative to the upper-left corner of the screen.
GetWindowText	LPCTSTR GetWindowText() const;
Get Window Text	Retrieves the title or text associated with the window.
GetWindowTextLength	<pre>int GetWindowTextLength() const;</pre>
	Retrieves the length, in characters, of the specified window's text.
HiliteMenuItem	BOOL HiliteMenuItem(CMenu* pMenu, UINT uItemHilite,
	Highlights or removes the highlighting from an item in a menu bar. This func
InvalidateRect	BOOL InvalidateRect(LPCRECT lpRect, BOOL bErase = T
invaluateRect	The InvalidateRect function adds a rectangle to the window's update region. The portion of the window's client area that must be redrawn.
	BOOL InvalidateRgn(CRgn* pRgn, BOOL bErase = TRUE)
InvalidateRgn	The InvalidateRgn function invalidates the client area within the specified regulate region of a window. The invalidated region, along with all other areas painting when the next WM_PAINT message occurs.
IsChild	BOOL IsChild(CWnd* pChild) const;
	Tests whether a window is a child window or descendant window of a specifi is the direct descendant of a specified parent window if that parent window is the chain of parent windows leads from the original overlapped or pop-up wi
IsDialogMossago	BOOL IsDialogMessage(LPMSG lpMsg) const;
IsDialogMessage	Determines whether a message is intended for the specified dialog box and, i
	<pre>UINT IsDlgButtonChecked(int nIDButton) const;</pre>
IsDlgButtonChecked	Determines whether a button control has a check mark next to it or whether a grayed, checked, or neither.
IsIconic	BOOL IsIconic() const;
	Determines whether the window is minimized (iconic). This function is not s
IsWindow	BOOL IsWindow() const;
	Determines whether the window handle identifies an existing window.
IsWindowEnabled	BOOL IsWindowEnabled() const;
	Determines whether the window is enabled for mouse and keyboard input.

IsWindowVisible	BOOL IsWindowVisible() const;
	Retrieves the visibility state of the window.
IsZoomed	BOOL IsZoomed() const;
	Determines whether the window is maximized. This function is not supported o
KillTimer	BOOL KillTimer(UINT_PTR uIDEvent) const;
	Destroys the specified timer.
LockWindowUpdate	B00L LockWindowUpdate() const;
	Disables or enables drawing in the window. Only one window can be locked at a supported on WinCE.
	<pre>void MapWindowPoints(CWnd* pWndTo, POINT&amp; pt) const;</pre>
	<pre>void MapWindowPoints(CWnd* pWndTo, RECT&amp; rc) const;</pre>
MapWindowPoints	<pre>void MapWindowPoints(CWnd* pWndTo, LPPOINT ptArray, U</pre>
	Converts (maps) a set of points from a coordinate space relative to one window another window.
M	<pre>int MessageBox(LPCTSTR lpText, LPCTSTR lpCaption, UIN</pre>
MessageBox	Creates, displays, and operates a message box. The message box contains an app title, plus any combination of predefined icons and push buttons.
MoveWindow	BOOL MoveWindow(int x, int y, int nWidth, int nHeight BOOL bRepaint = TRUE) const;
MoveWindow	B00L MoveWindow(const RECT& rc, B00L bRepaint = TRUE)
	The MoveWindow function changes the position and dimensions of the window
OpenIcon	BOOL OpenIcon() const;
	Restores a minimized (iconic) window to its previous size and position, and ther
PostMessage	BOOL PostMessage(UINT uMsg, WPARAM wParam = 0L, LPARA
	BOOL PostMessage(HWND hWnd, UINT uMsg, WPARAM wParam,
	The PostMessage function places (posts) a message in the message queue associ the window and returns without waiting for the thread to process the message.
	<pre>void Print(CDC* pDC, DWORD dwFlags) const;</pre>
Print	Requests that the window draw itself in the specified device context, most comm This function is not supported on WinCE.

RedrawWindow	BOOL RedrawWindow(LPCRECT lpRectUpdate = NULL, CRgn* UINT flags = RDW_INVALIDATE   RDW_U RDW_ERASE   RDW_ALLCHILDREN) const;	
	This function updates the specified rectangle or region in a window's client are	
ReleaseDC	<pre>int ReleaseDC(CDC* pDC) const;</pre>	
	Releases a device context (DC), freeing it for use by other applications.	
	B00L ScreenToClient(P0INT& Point) const;	
ScreenToClient	B00L ScreenToClient(RECT& rc) const;	
	Converts the screen coordinates of a specified RECT or points on the screen to	
ScrollWindow	BOOL ScrollWindow(int XAmount, int YAmount, LPCRECT prcClip) const;	
	Scrolls the contents of the specified window's client area. This function is not s applications should use the ScrollWindowEx function.	
ScrollWindowEx	<pre>int ScrollWindowEx(int dx, int dy, LPCRECT prcScro CRgn* prgnUpdate, LPRECT prcUpd</pre>	
	Scrolls the contents of the specified window's client area. This function is not s	
SendDlgItemMessage	LRESULT SendDlgItemMessage(int nIDDlgItem, UINT Msg, WPARAM wParam, LPARAM lPa	
	Sends a message to the specified control in a dialog box.	
SendMessage	LRESULT SendMessage(UINT uMsg, WPARAM wParam = 0L, LPARAM lParam = 0L) const;	
	LRESULT SendMessage(HWND hWnd, UINT uMsg, WPARAM wPa LPARAM lParam) const;	
	The SendMessage function sends the specified message to a window or windo for the window and does not return until the window procedure has processed	
SendNotifyMessage	BOOL SendNotifyMessage(UINT Msg, WPARAM wParam, LPAR	
	Sends the specified message to a window or windows. If the window was creat SendNotifyMessage calls the window procedure for the window and does not has processed the message. If the window was created by a different thread, Se message to the window procedure and returns immediately; it does not wait for processing the message.	
SetActiveWindow	CWnd* SetActiveWindow() const;	

	Activates a window. The window will be brought into the foreground (top foreground when the system activates the window.	
	CWnd* SetCapture() const;	
SetCapture	Sets the mouse capture to the window belonging to the current thread. SetC when the mouse is over the capturing window, or when the mouse button w the capturing window and the button is still down. Only one window at a time capturing window and the button is still down.	
	ULONG_PTR SetClassLongPtr(int nIndex, LONG_PTR dwN	
SetClassLongPtr	Replaces the specified value at the specified offset in the extra class memory for the class to which the window belongs.	
SetDlgItemInt	BOOL SetDlgItemInt(int nIDDlgItem, UINT uValue, BO	
	Sets the text of a control in a dialog box to the string representation of a spe	
SetDlgItemText	BOOL SetDlgItemText(int nIDDlgItem, LPCTSTR lpStri	
	Sets the title or text of a control in a dialog box.	
SetFocus	CWnd* SetFocus() const;	
	Sets the keyboard focus to the window.	
SetFont	<pre>void SetFont(CFont pFont, B00L bRedraw) const;;</pre>	
	Specifies the font that a window will use when drawing text.	
SetForegroundWindow	BOOL SetForegroundWindow() const;	
	Puts the thread that created the window into the foreground and activates th directed to the window, and various visual cues are changed for the user. Th priority to the thread that created the foreground window than it does to oth	
SetIcon	HICON SetIcon(HICON hIcon, BOOL bBigIcon) const;	
Selection	Associates a new large or small icon with the window	
SetMenu	BOOL SetMenu(CMenu* pMenu) const;	
	Assigns a new menu to the window. This function is not supported on WinO	
SetParent	CWnd* SetParent(CWnd* pWndParent) const;	
	Changes the parent window of the specified child window.	
	B00L SetRedraw(B00L bRedraw = TRUE) const;	
SetRedraw	This function allows changes in the window to be redrawn or prevents chan redrawn, by sending a WM_SETREDRAW message.	

SetScrollInfo	<pre>int SetScrollInfo(int fnBar, const SCROLLINFO&amp; si,</pre>
	Sets the parameters of a scroll bar, including the minimum and maximum scr the position of the scroll box (thumb). The function also redraws the scroll ba
	<pre>int SetScrollPos(int nBar, int nPos, BOOL bRedraw)</pre>
SetScrollPos	Sets the position of the scroll box (thumb) in the specified scroll bar and, if r reflect the new position of the scroll box. New applications should use the Se is not supported on WinCE.
SetScrollRange	BOOL SetScrollRange(int nBar, int nMinPos, int nMax BOOL bRedraw) const;
эегэсгонкапде	Sets the minimum and maximum scroll box positions for the specified scroll the SetScrollInfo function. This function is not supported on WinCE.
SetTimer	<pre>UINT_PTR SetTimer(UINT_PTR nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc) const;</pre>
	Creates a timer with the specified time-out value.
	LONG_PTR SetWindowLongPtr(int nIndex, LONG_PTR dwNe
SetWindowLongPtr	Changes an attribute of the window. The function also sets a value at the spe memory.
	B00L SetWindowPlacement(const WINDOWPLACEMENT& wndp
SetWindowPlacement	Sets the show state and the restored, minimized, and maximized positions of supported on WinCE.
	BOOL SetWindowPos(const CWnd* pInsertAfter, int x, int cx, int cy, UINT uFlags) cons
SetWindowPos	BOOL SetWindowPos(const CWnd* pInsertAfter, const R UINT uFlags) const;
	Changes the size, position, and Z order of a child, pop-up, or top-level window windows are ordered according to their appearance on the screen. The topmorand is the first window in the Z order. The pInsertAfter can one of: &wndTop, &wndTopMost, &wndBottom, or &
	int SetWindowRgn(CRgn* pRgn, B00L bRedraw = TRUE) c
SetWindowRgn	Sets the window region of a window. The window region determines the are system permits drawing. The system does not display any portion of a windo region. The system owns the region after a successful call to SetWindowRgr
	B00L SetWindowText(LPCTSTR lpString) const;
SetWindowText	Changes the text of the window's title bar (if it has one). If the window is a c

	changed.
SetWindowTheme	<pre>HRESULT SetWindowTheme(LPCWSTR pszSubAppName, LPCWSTR</pre>
	Causes a window to use a different set of visual style information (XP Theme) th
ShowOwnedPopups	BOOL ShowOwnedPopups(BOOL fShow) const;
ShowOwneuropups	Shows or hides the all the pop-up windows owned by this window.
ShowScrollBar	BOOL ShowScrollBar(int nBar, BOOL bShow) const;
SHOWSCIUIIDai	Shows or hides the specified scroll bar. This function is not supported on WinCI
ShowWindow	B00L ShowWindow(int nCmdShow = SW_SH0WN0RMAL) const;
Show whiteow	Sets the window's show state.
ShowWindowAsync	B00L ShowWindowAsync(int nCmdShow) const;
Show white was ync	The ShowWindow function sets the window's show state created in a different tl
UpdateWindow	BOOL UpdateWindow() const;
	Updates the client area of the window by sending a WM_PAINT message to the region is not empty. If the update region is empty, no message is sent.
ValidateRect	BOOL ValidateRect(LPCRECT prc) const;
	Validates the client area within a rectangle by removing the rectangle from the u
ValidateRgn	B00L ValidateRgn(CRgn* pRgn) const;
	Validates the client area within a region by removing the region from the current
WindowFromDC	CWnd* WindowFromDC(CDC* pDC) const;
	Retrieves a pointer to the window associated with the specified device context. temporary, so don't save it for later use.
WindowFromPoint	<pre>static CWnd* WindowFromPoint(POINT pt);</pre>
vindowi ionii onii	Retrieves the window that contains the specified point (in screen coordinates).

# Remarks

Refer to the <u>Simple Window</u> section for information on how to use this class to create a window and handle its messages.

Win32/64 support Yes
WinCE support Yes

# Shared\_Ptr Class

# Description

A Shared\_Ptr is a smart pointer that can be used in a STL container, such as vector. It mimics the behaviour of "share\_ptr" which will be included in the next standard of C++.

# Shared\_Ptr Members

Shared_Ptr	Constructor for the Shared_Ptr.
get	Returns the stored pointer.
swap	Exchanges the contents of two smart pointers.

unique Returns true if the use\_count equals 1. uSe\_COUNT Returns the number of Shared\_Ptr objects. operator = Assigns a pointer to the Shared\_Ptr. operator -> Returns the stored pointer. operator \* Returns the dereferenced stored pointer. operator == Compares the dereferenced stored pointers. Returns true if lhs == rhs. operator != Compares the dereferenced stored pointers. Returns true if lhs != rhs. operator < Compares the dereferenced stored pointers. Returns true if lhs < rhs. operator > Compares the dereferenced stored pointers. Returns true if lhs > rhs.

## Remarks

Shared\_Ptr wraps a reference-counted smart pointer around a dynamically allocated object. Unlike auto\_ptr, the Shared\_Ptr can be used as a smart pointer for objects stored in containers like std::vector. Do not use Shared\_Ptr (or shared\_ptr or auto\_ptr) for dynamically allocated arrays. See below for advice on how to wrap dynamically allocated arrays in a vector.

The next standard of C++ will also contain a shared\_ptr. Some modern compilers already have a shared\_ptr available as std::tr1::shared\_ptr. If your compiler already provides a shared\_ptr, or if you have Boost, you should use that smart pointer instead. This class has been provided for those users who don't have easy access to an "official" shared\_ptr. Note that this class is "Shared\_Ptr", a slightly different name to the future "shared\_ptr" to avoid naming conflicts.

Advantages of Shared\_Ptr (or shared\_ptr where available):

• Shared\_Ptr can be safely copied. This makes them suitable for containers.

- Shared\_Ptr automatically calls delete for the wrapped pointer when its last copy goes out of scope.
- Shared\_Ptr simplifies execution safety.

Without smart pointers, it can be quite challenging to ensure that every dynamically allocated pointer (i.e. use of new) is deleted in the event of all possible exceptions. In addition to the exceptions we throw ourselves, "new" itself will throw an exception it it fails, as does the STL (Standard Template Library which includes vector and string). Without smart pointers we often need to resort to additional try/catch blocks simply to avoid memory leaks when exception occur.

Examples of declaring a Shared\_Ptr:

Shared\_Ptr<CWnd> w1(new CWnd); Shared\_Ptr<CWnd> w1 = new CWnd; typedef Shared\_Ptr<CWnd> CWndPtr; CWndPtr w1 = new CWnd; typedef Shared\_Ptr<CWnd> CWndPtr; CWndPtr w1(new CWnd);

Examples using a Shared\_Ptr in a vector:

```
typedef Shared_Ptr<CWnd> CWndPtr;
std::vector<CWndPtr> MyVector;
MyVector.push_back(new CWnd);
```

```
typedef Shared_Ptr<CWnd> CWndPtr;
CWnd* pWnd = new CWnd;
std::vector<CWndPtr> MyVector;
MyVector.push_back(pWnd);
```

Header file	shared_ptr.h
Win32/64	Yes
support	
WinCE support	Yes

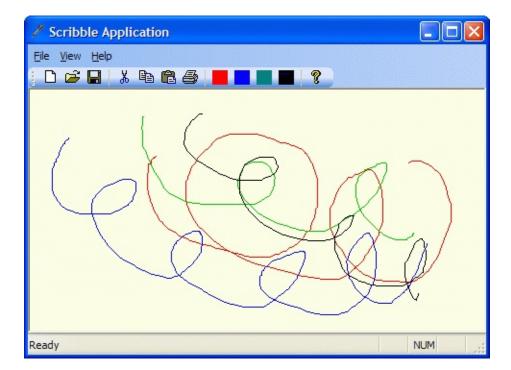
# **Tutorials**

#### Menu of tutorials

- Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance
- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing

Tutorial 10: Finishing Touches

This set of tutorials will demonstrate how to create a scribble application, capable of drawing on a window using a mouse. It starts with a simple window and ends with a frame based scribble application which supports the loading and saving saving its data to a file, and printing the contents of the window. The code for these tutorials can be found in the tutorials directory.



The application produced by completing the tutorial looks like this.

Choose from the set of links in the Menu of Tutorials to start.

# **Frequently Asked Questions (FAQ)**

How do I install Win32++? How do I start a project of my own? Why does Win32++ use a map for HWNDs and CWnd\*s? Why does Win32++ use TLS (Thread Local Storage) during window creation? How do I create a window that is initially hidden? How do I create a window that is initially minimised or maximised? How should I end an application? How do I access one CWnd from within another? How do I avoid automatically adding the Win32++ namespace to the global namespace? How can I tell what type of window a CWnd or HWND is? Should I use PostMessage or SendMessage, and what is the difference? How do I get a WM\_LBUTTONDBLCLK message for left button double clicks? How do add icons to menu items? How do I theme my popup menu? How do I create a Wizard? How do I avoid memory leaks? How do I test for memory leaks? How do I avoid GDI resource leaks? How do I check if my program is leaking memory or GDI resources? Why does my program assert? Why does my program throw an exception? Why are none of my resources working? How do I avoid flicker in my program? Why does the desktop flicker when I run my program? How do I get the Desktop's window handle? When are pointers from FromHandle deleted, and how do I use them safely? How do I do Idle processing? When do I need to use detach for a CDC? How do I use XP themes? How do I override the CToolBar or CStatusBar class for frames? How do I implement a wait cursor? How do I create a vertical ToolBar?

#### How do I install Win32++?

Obtain a copy of Win32++ from sourceforge, and extract the zip file into a directory of your choosing. The directory you choose might be within your documents folder. Be sure to retain the directory structure of Win32++ when extracting files from the zip archive. To download a copy of Win32++ proceed to the project's hosting at SourceForge <u>here</u>.

#### How do I start a project of my own?

Win32++ contains a set of "NewProject" files in the "new projects" folder for this purpose. It might prove easier though to make a copy of one of the samples that ship with Win32++, and use that as your starting point.

# Why does Win32++ use a map for HWNDs and CWnd\*s rather than CreateWindow's lpParam when creating the window?

It would be possible to store the CWnd pointer in the lpParam during window creation, but this has its limitations. MDI windows, for example, use lpParam for its own purpose during window creation.

Using a map to store the CWnd pointer and HWND also support the framework's use of FromHandle to retrieve the CWnd pointer associated with a HWND.

#### Why does Win32++ use TLS (Thread Local Storage) during window creation?

Win32++ supports the creation of windows in different threads. When several windows are created in different threads simultaneously, TLS keeps the code thread safe.

#### How do I create a window that is initially hidden?

Windows created by Win32++ are visible by default. To create a window that is initially hidden, override PreCreate and specify a windows style that does not include WS\_VISIBLE style as follows:

```
void CMyWin::PreCreate(CREATESTRUCT &cs)
{
    // Sets the CREATESTRUCT parameters prior to window creation
    cs.style = 0; // No WS_VISIBLE style is set
}
```

Use ShowWindow when you are ready to display the window.

#### How do I create a window that is initially minimised or maximised?

Create a window that is initially hidden, and then use ShowWindow to display it as minimised or maximised by specifying the appropriate value for nCmdShow e.g SW\_SHOWMAXIMIZED or SW\_SHOWMINIMIZED.

There are window styles called WS\_MAXIMIZE and WS\_ICONIC which can be specified when creating windows, but these don't apply to top level windows. They can be used for MDI child windows.

#### How should I end an application?

In general, the best way to end an application is to post a WM\_CLOSE to the top level window. This gives the application an opportunity to save any settings and clean up objects before destroying the main window and ending the program. The main window should then issue a PostQuitMessage when its window is destroyed. Win32++ issues the PostQuitMessage for us when a window inherited from CFrame is destroyed.

#### How do I access one CWnd object from within another?

Child windows for the CWnd are normally Class members of the CWnd. We should access the CWnd of child windows through these class members.

If the HWND of the window is known, we can use FromHandle to get the CWnd associated with it. If the relationship between the windows are known, we can use functions like GetParent and GetAncestor to access other windows.

Finally, we can use GetApp to get a pointer to our CWinApp class. The class you inherit from CWinApp could have a member that returns a pointer to the top level window it created. You could use that to step to the window you need.

For example.

Suppose CFrameApp inherits from CWinApp, and has CMainFrame as one of its class members. CFrameApp could have a member function that returns a pointer to CMainFrame. To access the frames status bar from anywhere we could use:

```
CFrameApp* pApp = (CFrameApp*)GetApp();
CStatusBar* pStatus = pApp->GetMainFrame()->GetStatusBar();
```

Note that a child window wouldn't normally call the functions of a parent CWnd directly. In most circumstance this represents poor program design. It is usually more appropriate for the child CWnd to send a message or notification to the parent and let the parent decide what to do with it. The message could be a user defined message or notification.

#### How do I avoid automatically adding the Win32xx namespace to the global namespace?

If the NO\_USING\_NAMESPACE macro is defined, the Win32xx namespace will not be added to the global namespace.

### How can I tell what type of window a CWnd or HWND is?

We can use dynamic cast to identify the type of CWnd our CWnd\* points to. For example:

```
if ( dynamic_cast<CToolBar*>(pWnd) )
{
    TRACE( _T("pWnd is a CToolBar*\n") );
}
```

We can also look at the class name of the window to identify its type.

```
if (lstrcmp(GetParent()->GetClassName(), _T("ReBarWindow32")) == 0)
{
    TRACE( _T("The Parent window is a ReBar control\n") );
}
```

#### Should I use PostMessage or SendMessage, and what is the difference?

The SendMessage function calls the window procedure for the specified window and does not return until the window procedure has processed the message. The PostMessage function places (posts) a message in the message queue associated with the thread that created the window and returns without waiting for the thread to process the message.

Normally we would use SendMessage for windows in the current thread, and PostMessage for windows in a different thread. In multi-threaded applications it is usually undesirable to hold up a thread with a window, waiting for a window in a different thread to respond.

#### How do I get a WM\_LBUTTONDBLCLK message for left button double clicks?

The CS\_DBLCLKS class style allows the window to send the WM\_LBUTTONDBLCLK message in response to a double left button click. Use the PreRegisterClass to register a window class with the CS\_DBLCLKS style.

#### How do I add icons to menu items?

There are two functions used to add icons to menu items. The AddMenuIcons functions adds a group of icons to menu items from a bitmap, and AddMenuIcon adds an individual icon.

#### How do I theme my popup menu?

The CWnd we specify in TrackPopupMenu or TrackPopupMenuEx receives notifications from the menu. When we specify CFrame's CWnd in these functions, CFrame will perform the drawing of the menu for us.

#### How do I create a Wizard?

A wizard is a type of property sheet. Wizards are designed to present pages one at a time in a sequence that is controlled by the application. Refer to the propertysheet sample for a demonstration of how to create a wizard.

#### How do I avoid memory leaks?

Memory leaks are generated when memory is allocated from the heap, and not returned properly. C++ code we should always allocate dynamic memory by using new, and delete it with delete, or delete[] for arrays.

It is wise to store the pointer returned by new in a smart pointer. Shared\_Ptr is a smart pointer provided by the Win32++ framework for this purpose. It can be safely stored in a vector. Storing the pointer in a smart pointer like Shared\_Ptr eliminates the need for delete. The Shared\_Ptr deletes the pointer for us when it goes out of scope. Storing the pointer in a smart pointer also helps ensure the code doesn't leak memory in the event of an exception, without additional of try catch blocks.

### How do I test for memory leaks?

If you are using a Microsoft compiler, Visual Leak Detector can be used to identify memory leaks. Visual Leak Detector is a free utility. It can be downloaded from <u>https://vld.codeplex.com/</u>.

#### How do I avoid GDI resource leaks?

GDI resources are limited. If the code leaks GDI resources, it will ultimately fail when no more GDI resources are available. The best way to avoid GDI leaks is to use the CDC and GDI object classes provided with Win32++. These automatically release the device contexts and GDI resources.

#### How do I check if my program is leaking memory or GDI resources?

The window's taskmgr.exe is a very useful tool for identifying memory leaks and GDI resource leaks. It ships with windows and can be configured to display extra columns including GDI objects and working set memory.

If the amount of memory used or number of GDI objects increase without limit when we do things like resize the window, we have a leak.

#### Why does my program assert?

Asserts are only triggered in debug mode. They are always caused by programming errors, so the cause of the assert should be identified and fixed.

Win32++ will assert to warn that the library has been used improperly. The following sorts of things will cause Win32++ to assert.

- A CWnd creating a window, when it already has a window assigned to it. If this is intended, the current window should be destroyed first.
- A CWnd performing a window operation (such as ShowWindow) before the window is created.

# • Attaching a GDI handle (e.g. a HPEN) to a CGDIObject when it already has a handle attached. If this is intended, the current handle should be deleted or detached first.

Remember asserts our our friend. They give us early warning of bugs that might otherwise go unnoticed or be difficult to track down. They help us build robust, reliable code. For this reason, they should be used liberally in the code we write.

#### Why does my program throw an exception?

Exceptions should indicate that an unexpected error occurred. They may not necessarily indicate a programming error, but programming errors can certainly cause exceptions. If the exception is "handled", the program will continue to run. If the exception is unhandled, the program will abort. For example, C++ will generate an exception if an attempt to allocate memory dynamically using "new" fails.

Win32++ generates an exception when an attempt to create a window fails (in which case it will attempt to indicate why). It will also generate and handle an exception if an attempt to save values in the registry fails.

You may choose to generate and handle exceptions in your own code. As the name suggests though, the generation of exceptions should be the exception, not the rule.

#### Why are none of my resources working?

Resources are defined in a resource script file, often called "resource.rc" This file needs to be added to the set of files compiled by our compiler.

#### How do I avoid flicker in my program?

Flicker is an annoying visual effect caused when a window is rapidly redrawn differently. It may show up when we resize the window. Depending on the cause of the flicker, these techniques might help.

- Use double buffering. With this technique we do all the drawing on a memory device context. When the drawing is complete, copy this to the window with BitBlt so we only draw to the window once.
- Suppress the redrawing of the window background by handling OnEraseBkgnd.
- Use DeferWindowPos to reposition a set of windows simultaneously.
- Avoid invalidating the window unnecessarily.
- Use of SetRedraw to turn window drawing off and on.

#### Why does the desktop flicker when I run my program?

Some functions, including InvalidateRect can use NULL as the handle for the desktop's window. If we perform an InvalidateRect on a NULL HWND, this will invalidate the desktop, forcing it to redraw.

#### How do I get the Desktop's window handle?

There are a few functions, such as GetDC and InvalidateRect that can use NULL as the HWND for the desktop. NULL isn't a real window handle however, and IsWindow(NULL) will return FALSE. The Windows API function ::GetDesktopWindow returns the handle to the desktop window, and Win32++'s GetDesktopWindow function returns a CWnd pointer to the desktop window.

#### When are the pointers from FromHandle deleted, and how do I use these function safely?

The FromHandle functions can return pointers to CWnd, CMenu, CDC, CImageList and CGDIObject when provided with the handle. If we use FromHandle to get the CWnd\* from a HWND, for example, the framework checks the CWnd map to see if a CWnd for the HWND already exists. If the CWnd already exists, its pointer is returned. If the CWnd doesn't already exist, a temporary CWnd is created and its pointer is returned. Pointers for other types of objects behave in the same way.

The pointer returned by FromHandle might point to a temporary object. Temporary objects are deleted automatically some time after

this current message is processed, and shouldn't be saved for later use. We should assume the pointer will be deleted when the framework fetches the next message from the message queue and processes it.

In practical terms, it is quite safe to copy the pointer returned by FromHandle and store it in a local variable in a function and use it there. The pointer will not be deleted until after the function completes. It would NOT be safe to store the pointer returned by FromHandle in a global variable, or a class member variable and expect to be able to use it sometime later.

#### How do I do Idle processing?

OnIdle will be called repetitively until it returns FALSE while there are no messages queued, incrementing loount each time it is called. To implement idle processing, override CWinApp::OnIdle. Larger tasks will need to be split into chunks to allow the application to respond promptly to keyboard and mouse input. For tasks that have been split into chunks, do the first chunk when loount is 0 and return TRUE, the second when loount is 1 and return TRUE etc. When all chunks are done, OnIdle should return FALSE. The framework doesn't use OnIdle internally, so there is no need to call CWinApp::OnIdle when when overriding OnIdle.

OnIdle is not suitable for tasks that need to be performed at regular intervals. Use a timer for those.

Its important to remember that idle processing was introduced for 16 bit windows because it wasn't capable of running multiple threads. In 32 bit and 64 bit windows, worker threads provide a better way of performing background tasks without affecting the user's interaction with the application. Having said that, idle processing can prove simpler than threads for very small tasks. While larger idle tasks can be split up into chunks and performed during different cycles of the OnIdle processing, a worker thread would do this more efficiently, and would probably be simpler to implement.

#### When do I need to use detach for a CDC?

There are some HDCs we shouldn't destroy. These are the HDCs that windows operating creates for us, and provides them in messages such as WM\_PAINT, WM\_ERASEBKGND, NM\_CUSTOMDRAW etc. If we choose to attach those HDCs to one of our CDCs we need to detach them before the CDC goes out of scope. In general, if we create the HDC, we are responsible for destroying it, but if the operating system creates the HDC for us, it is responsible for destroying it.

It can be be more convenient to use FromHandle to assign a temporary CDC for those HDCs provided to us by the operating system. Temporary CDCs don't destroy the HDC they manage when their destructor is called, so this eliminates the need to detach the HDC. For example, the Win32++ framework uses FromHandle to assign a temporary CDC to the HDC provided by the NM\_CUSTOMDRAW notification when performing custom drawing.

#### How do I use XP themes?

1

In order to use XP themes, our application must contain a manifest. We can specify a manifest section in our resource script (resource.rc) as follows:

### 

NI\_II

RT MANIFEST

"res/Win32++.manifest"

The manifest file supplied with the samples is compatible with both 32 bit and 64 bit windows.

Older compilers such as visual studio 6 and Borlands C++ compiler version 5.5 may not have a definition for RT\_MANIFEST. RT\_MANIFEST is defined as MAKEINTRESOURCE(24).

#### How do I override the CToolBar or CStatusBar class for frames?

If you should need to modify the CToolBar class that CFrame uses, you can do the following.

Inherit your new ToolBar from the CToolBar class. We will call the new ToolBar class CMyToolBar.
 Add a CMyToolBar member variable to CMainFrame called MyToolBar.
 Override CFrame's GetToolBar function in CMainFrame as follows:

# virtual CToolBar\* GetToolBar() const { return const\_cast<CMyToolBar\*>(&m;\_MyToolBar);

}

CMainFrame will now use your CMyToolBar class in place of CToolBar. You could use the same technique for overriding CStatusBar.

#### How do I implement a wait cursor?

SetCurser is the Windows API function which sets the cursor. By default, windows will set the cursor back to the one defined in the window class each time the mouse is moved. We can handle the WM\_SETCURSOR message to determine which cursor should be displayed when the mouse is moved.

We can implement a wait cursor like this:

```
void CView::DoWaitCursor(BOOL bWait)
Ł
  m bWait = bWait;
                   // m bWait is a member variable
 if (m bWait)
    SetCursor(LoadCursor(NULL, IDC WAIT));
  else
    SetCursor(LoadCursor(NULL, IDC ARROW));
}
LRESULT CView::WndProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
  switch (uMsg)
  {
  case WM SETCURSOR:
    if (m bWait)
    {
        SetCursor(LoadCursor(NULL, IDC WAIT));
        // Prevent setting the cursor back to default
        return 0L;
    }
    else
      SetCursor(LoadCursor(NULL, IDC_ARROW));
    break;
  }
  // Do default processing for other messages
  return WndProcDefault(uMsg, wParam, lParam);
}
```

#### How do I create a vertical ToolBar?

There are two rather similar methods for achieving this:

Method 1:

- Define a toolbar as you would for a horizontal toolbar
- Add the CCS\_LEFT (or CCS\_VERT or CCS\_RIGHT styles)

### • Add the TBSTATE\_WRAP style to each ToolBar button.

The ToolBar will be automatically placed on the side of the parent window, specified by the style (e.g CCS\_LEFT).

Method 2:

- Define a toolbar as you would for a horizontal toolbar
- Add the CCS\_LEFT (or CCS\_VERT or CCS\_RIGHT styles)
- Add the TBSTATE\_WRAP style to each ToolBar button.
- Position the ToolBar manually using SetWindowPos

Note:

- Method 2 is required for vertical ToolBars within ReBars
- CCS\_NODIVIDER can be added to suppress the drawing of the ToolBar's divider
- Use DeferWindowPos to reposition a group of windows simultaneously.

## **Contacting the Author**

Anyone with a question or comment relating to Win32++ are encouraged to post these in one of the forums <u>here</u> at SourceForge. You will find forums for support requests, feature requests, bug reports and general comments. I monitor the forums regularly, and am normally able to post a reply the next day.

If you would prefer to send me a direct email, you can contact me on <u>dnash@bigpond.net.au</u>

## **Some Useful Links**

### **Free Compilers and Tools**

<u>Dev-C++</u>	Dev-C++ is a Integrated Development Environment (IDE) for C/C++. It includes the MinGW port of GCC as it's compiler.
<u>Code::Blocks</u>	Code::Blocks is an IDE for C/C++. It includes the MinGW port of GCC, and also supports the MS Visual Toolkit, and the free command line compiler from Borland.
<u>Microsoft Visual C++</u> 2013 <u>Express</u>	A free C++ Win32 compiler and IDE from Microsoft.
Borland C++ Builder	A free C++ Win32 compiler from Borland. You will need to register before downloading.
<u>ResEdit</u>	A free Resource Editor for Win32 programs.
<u>GDIView</u>	A utility that displays list a of GDI handles used by each process. It's useful for spotting GDI leaks.
<u>Visual Leak Detector</u>	An open-source memory leak detection system for Visual C++.

### **Tutorials and References**

<u>MSDN</u>	The Microsoft Developers Network. Microsoft's online reference.
<u>MS Platform SDK</u>	Provides the documentation, samples, header files, libraries, and tools designed to help you develop Windows applications (XP and below).
MS Windows SDK	Provides the documentation, samples, header files, libraries, and tools designed to help you develop Windows applications (Vista and above).
<u>MS DirectX SDK</u>	Contains the additional header files and libraries for DirectX programming.
	SourceForge hosts a large collection of open source

<u>SourceForge</u>	projects.
Code Project	Forums and Code samples for software developers.

<u>Cfanatic</u> A forum for C, Win32, C++, MFC, C# developers. <u>Reliable Software</u> A Win32 tutorial for advanced C++ users.

the Forger A Min22 tyterial for C + +

the Forger A Win32 tutorial for C++.

<u>FunctionX</u> A Win32 programming reference, by FunctionX.

<u>Dev-C++ Resource Site</u> API references, tutorials, online resources,

documentation downloads, etc. about C/C++ and Win32 programming. <u>Thinking</u> <u>in C++</u> Thinking in C++ 2nd Edition. An excellent book on C++ by Bruce Eckel. Available Online. <u>Google</u> Internet Search Engine. <u>Wikipedia</u> Web based encyclopedia. <u>C++ FAQ</u> C++ Frequently Asked Questions

## **CMenuBar Class**

### Description

The CMenuBar class provides a menu inside a rebar control. This menu is based on a toolbar control, and CMenuBar inherits from CToolBar.

### **Member functions**

CMenuBar();

Constructor for CMenuBar.

### GetMenu

HMENU GetMenu() const;

Returns the handle to the menu displayed by the menubar. SetMenu

```
virtual void SetMenu(HMENU hMenu);
```

Sets the menu for the MenuBar.

### **Overridables**

OnCreate	<pre>virtual void OnCreate();</pre>
	Called when the window is created.
OnCustomDraw	<pre>virtual LRESULT OnCustomDraw(NMHDR* pNMHDR);</pre>
	Called to perform custom drawing.
OnKeyDown	<pre>virtual void OnKeyDown(WPARAM wParam, LPARAM lParam);</pre>
	Called when a key is pressed.
OnLButtonDown	<pre>virtual void OnLButtonDown(WPARAM wParam, LPARAM lParam)</pre>
	Called when the left mouse button is pressed.

OnLButtonUp	<pre>virtual void OnLButtonUp(WPARAM wParam, LPARAM lParam);</pre>
	Called when a key is released.
OnMenuChar	<pre>virtual void OnMenuChar(WPARAM wParam, LPARAM lParam);</pre>
Omvienucitai	Called when the user presses a menu mnemonic character that doesn't match any of mnemonics in the current menu.
OnManufacut	<pre>virtual BOOL OnMenuInput(UINT uMsg, WPARAM wParam, LPAR/</pre>
OnMenuInput	Called to process menu input.
OnMouseLeave	<pre>virtual void OnMouseLeave();</pre>
OIIIVIOUSELEave	Called when the cursor leave the client area of the window.
OnMouseMove	<pre>virtual void OnMouseMove(WPARAM wParam, LPARAM lParam);</pre>
	Called when the cursor moves.
OnNotifyReflect	virtual LRESULT OnNotifyReflect(WPARAM wParam, LPARAM lF
	Called in response to a notification from the menubar.
OnSysCommand	virtual void OnSysCommand(WPARAM wParam, LPARAM lParam);
	Called to processes keys when the Alt or F10 key is pressed.
OnWindowPosChanged	<pre>virtual void OnWindowPosChanged();</pre>
	Called when the windows size or position has changed.
PreCreate	<pre>virtual void PreCreate(CREATESTRUCT &amp;cs);</pre>
	Sets the window creating parameters.
PreRegisterClass	<pre>virtual void PreRegisterClass(WNDCLASS &amp;wc);</pre>
FIEREgisterClass	Sets the window class parameters.

### **Base class Members**

For base class members, refer to the members of <u>CToolBar</u>.

### Remarks

CMenuBar is used by <u>CFrame</u> and <u>CMDIFrame</u> to provide the menu.

### Summary Information

Header file	frame.h
Win32/64 support	Yes
WinCE support	No
Library required	Comctl32.lib

## **CReBar Class**

### Description

The class responsible for creating a ReBar control. ReBar controls act as containers for child windows. An application assigns child windows, which are often other controls, to a rebar control band. ReBar controls contain one or more bands, and each band can have any combination of a gripper bar, a bitmap, a text label, and a child window.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of rebar controls.

### **CReBar Members**

 CReBar
 CReBar();

 Constructor for CReBar.

 DeleteBand

 BOOL DeleteBand(const int nBand) const;

 Deletes a band from the rebar control. GetBand

 int GetBand(const HWND hWnd) const;

 Returns the band number given the band's window handle. GetBandBorders

 CRect GetBandBorders(int nBand) const;

 Retrieves the borders of a band. The result of this message can be used to calculate the usable area in a band. GetBandCount

 int GetBandCount() const;

 Retrieves the count of bands currently in the rebar control. GetBandInfo

 BOOL GetBandInfo(const int nBand, REBARBANDINFO& rbbi) const;

 Retrieves information about a specified band in a rebar control. GetBandRect

### CRect GetBandRect(int i) const;

Retrieves the bounding rectangle for a given band in a rebar control. GetBarHeight

UINT GetBarHeight() const;

Retrieves the height of the rebar control. GetRowCount

UINT GetRowCount() const;

Retrieves the number of rows of bands in a rebar control. GetRowHeight

int GetRowHeight(int nRow) const;

Retrieves the height of a specified row in a rebar control. GetSizeofRBBI

UINT GetSizeofRBBI() const;

Returns the correct value for sizeof(REBARBANDINFO). GetToolTips

CToolTip\* GetToolTips() const;

Retrieves the handle to any ToolTip control associated with the rebar control. HitTest

int HitTest(RBHITTESTINFO& rbht);

HWND HitTest(POINT pt);

Determines which portion of a rebar band is at a given point on the screen, if a rebar band exists at that point. **IDToIndex** 

int IDToIndex(UINT uBandID) const;

Converts a band identifier to a band index in a rebar control. InsertBand

BOOL InsertBand(const int nBand, REBARBANDINFO& rbbi) const;

Inserts a new band in a rebar control. IsBandVisible

B00L IsBandVisible(int nBand) const;

Returns TRUE if band is visible, otherwise FALSE MaximizeBand

void MaximizeBand(UINT uBand, BOOL fIdeal = FALSE);

Resizes a band in a rebar control to either its ideal or largest size. MinimizeBand

### void MinimizeBand(UINT uBand);

Resizes a band in a rebar control to its smallest size. MoveBand

BOOL MoveBand(UINT uFrom, UINT uTo);

Moves a band from one index to another. MoveBandsLeft

void MoveBandsLeft();

Repositions the bands so that they are moved to the left. ResizeBand

BOOL ResizeBand(const int nBand, const CSize& sz) const;

Changes the size of a rebar band. SetBandBitmap

BOOL SetBandBitmap(const int nBand, const HBITMAP hBackground) const;

Sets a bitmap to the ReBar control SetBandColor

BOOL SetBandColor(const int nBand, const COLORREF clrFore, const COLORREF clrBack) const;

Sets the colour of a ReBar band SetBandInfo

BOOL SetBandInfo(const int nBand, REBARBANDINFO& rbbi) const;

Sets characteristics of an existing band in the rebar control. SetBarInfo

B00L SetBarInfo(REBARINF0& rbi) const;

Sets the characteristics of a rebar control. ShowBand

BOOL ShowGripper(int nBand, BOOL fShow) const;

Shows or hides a band ShowGripper

BOOL ShowBand(int nBand, BOOL fShow) const;

Shows or hides a band's gripper. SizeToRect

B00L SizeToRect(CRect& rect) const;

Finds the best layout of the bands from the given rectangle.

**Base class Members** 

For base class members, refer to the members of <u>CWnd</u>.

### Remarks

Like all common controls, the rebar control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a rebar control.

The following code creates a ComboBox and adds it to the rebar.

```
void CMainFrame::AddListboxBand(int Listbox Height)
ł
  // Get the reference to the rebar object
  CReBar\& RB = GetReBar();
  ThemeReBar RBTheme = RB.GetReBarTheme();
  // Create the ComboboxEx window
  CREATESTRUCT cs = \{0\};
  cs.lpszClass = T("COMBOBOXEX32");
  cs.style = WS_VISIBLE | WS_CHILD | CBS_DROPDOWN;
  cs.cy = 100; // required to display list
  cs.hMenu = (HMENU)IDC COMBOBOXEX;
  m ComboboxEx.PreCreate(cs);
  m ComboboxEx.Create(GetReBar().GetHwnd());
  // Put the window in a new rebar band
  REBARBANDINFO rbbi = \{0\};
  rbbi.cbSize = GetSizeofRBBI();
  rbbi.fMask = RBBIM COLORS | RBBIM CHILDSIZE | RBBIM STYLE | RBBIM
  rbbi.cyMinChild = Listbox Height;
  rbbi.cyMaxChild = Listbox Height;
  rbbi.cxMinChild = 200;
  rbbi.fStyle = RBBS_BREAK | RBBS_VARIABLEHEIGHT | RBBS_GRIPPERALWAY
rbbi.clrFore = GetSysColor(COLOR_BTNTEXT);
rbbi.clrBack = RBTheme.clrBand1;
  rbbi.hwndChild = m ComboboxEx.GetHwnd();
  rbbi.lpText = T("Address");
  RB.InsertBand(-1, rbbi);
}
```

CRebar is used by <u>CFrame</u> to contain a <u>CMenuBar</u> and a <u>CToolBar</u>.

### **Summary Information**

Header file	rebar.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

## **CScrollBar Class**

### Description

The CScrollBar class is used to create and manage a scroll bar control. Scroll bars are used when the window isn't large enough to display all of its content, such as a bitmap or a document. Scroll bars can be used to bring into view the portions of the content that extend beyond the borders of the window.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of scroll bar controls.

### **CScrollBar Members**

CScrollBar();

Constructor for the CScrollBar.

### EnableScrollbar

BOOL EnableScrollBar( UINT nArrowFlags = ESB\_ENABLE\_BOTH ) const;

Enables or disables one or both scroll bar arrows. GetScrollInfo

BOOL GetScrollInfo(LPSCROLLINFO lpsi) const;

Retrieves information that the SCROLLINFO structure maintains about the scroll bar. GetScrollPos

int GetScrollPos() const;

Retrieves the current position of the scroll box. GetScrollRange

BOOL GetScrollRange(LPINT lpMinPos, LPINT lpMaxPos ) const;

Retrieves the current minimum and maximum scroll box positions. SetScrollInfo

BOOL SetScrollInfo(LPSCROLLINFO lpsi, BOOL bRedraw = TRUE ) const;

Sets the information that the SCROLLINFO structure maintains about the scroll bar. SetScrollPos

int SetScrollPos(int nPos, BOOL bRedraw) const;

Sets the position of the scroll box and, if requested, refraws the scroll bar to reflect the new position of the scroll box. SetScrollRange

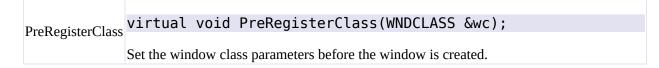
```
BOOL SetScrollRange( int nMinPos, int nMaxPos, BOOL bRedraw = TRUE ) con
```

Sets the minimum and maximum scroll box positions for the scroll bar. ShowScrollBar

```
BOOL ShowScrollBar(BOOL bShow) const;
```

Shows or hides the scroll bar.

### Overridables



### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

### Remarks

Like all common controls, the scroll bar control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a scroll bar control.

There are two ways to add scroll bars to a window.

- Create the window with the WS\_HSCROLL and/or WS\_VSCROLL styles.
- Create a horizontal and/or vertical scroll bar control as a child window we wish to scroll.

Use the SBS\_HORZ style to create a horizontal scroll bar control, and the SBS\_VERT style to create a vertical scroll bar control.

### **Summary Information**

Header file	controls.h
Win32/64	Yes
support	165

WinCE support Yes

## **CStatusBar Class**

### Description

The class responsible for creating and managing a status bar. Status bars display status information in a horizontal window at the bottom of an application window. Status bars are often divided into parts, and each part displays different status information.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of status bar controls.

### **CStatusBar Members:**

CStatusBar	CStatusBar();	
	Constructor for CStatusBar.	
CreatePar	rts	
BOOL Create	Parts(int iParts, const int iPaneWidths[]) const;	
Creates one or n	nore status bar parts. GetPartIcon	
HICON GetPa	rtIcon(int iPart);	
Retrieves the icc	on for a part in the status bar. GetPartRect	
CRect GetPa	rtRect(int iPart);	
Retrieves the bo	unding rectangle of a part in the status bar. $\operatorname{GetParts}$	
<pre>int GetParts();</pre>		
Retrieves a count of the parts in the status bar. GetPartText		
CString GetPartText(int iPart) const;		
Retrieves the text from a part in the status bar. SetPartIcon		
BOOL SetPar	tIcon(int iPart, HICON hIcon);	

Sets the icon for a part in the status bar.  $\ensuremath{SetPartText}$ 

BOOL SetPartText(int iPart, LPCTSTR szText, UINT Style = 0) const;

Sets the text for a part in the status bar. SetPartWidth

BOOL SetPartWidth(int iPart, int iWidth) const;

Sets the width of an existing status bar pane, or creates a new pane with the specified width. **SetSimple** 

```
void SetSimple(BOOL fSimple = TRUE);
```

Specifies whether a status window displays simple text or displays all window parts set by a previous SB\_SETPARTS message.

### **Base class Members**

For base class members, refer to the members of <u>CWnd</u>.

### Remarks

Like all common controls, the status control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a status control.

The following code segment demonstrates how to create four panes in a status bar, and add some text.

```
void CMainFrame::SetStatusText()
{
    if (::IsWindow(GetStatusBar()))
    {
        // Get the coordinates of the frame window's client area.
        CRect rcClient = GetClientRect();
        // width = max(300, rcClient.right)
        int width = (300 > rcClient.right) ? 300 : rcClient.right;
        int iPaneWidths[] = {width - 110, width - 80, width - 50, width - 20
        if (m_bShowIndicatorStatus)
        {
            // Create 4 panes
            GetStatusBar().CreateParts(4, iPaneWidths);
        }
    }
}
```

```
// Or you could create the 4 panes this way
// GetStatusBar().SetPartWidth(0, width - 110);
// GetStatusBar().SetPartWidth(1, 30);
// GetStatusBar().SetPartWidth(2, 30);
// GetStatusBar().SetPartWidth(3, 30);
SetStatusIndicators();
}
// Place text in the 1st pane
GetStatusBar().SetPartText(0, _T("Some Text"));
}
```

### **Summary Information**

Header file	statusbar.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

## **CToolBar Class**

### Description

This is the class responsible for creating and managing a toolbar. A toolbar is a control window that contains one or more buttons. Each button, when clicked by a user, sends a command message to the parent window. Typically, the buttons in a toolbar correspond to items in the application's menu, providing an additional and more direct way for the user to access an application's commands.

Refer to the documentation that ships with the Microsoft Windows Software Development Kit for more information on the use of toolbar controls.

### **CToolBar Members**

CToolBar	CToolBar();		
	Constructor for CToolBar.		
AddBitm	ар		
virtual int	AddBitmap(UINT ToolBarID);		
Adds a bitmap t	o the toolbar. See also SetImages. $\operatorname{AddButton}$		
virtual BOOL AddButton(UINT nID, BOOL bEnabled = TRUE);			
Adds one buttor	to the toolbar. AddButtons		
BOOL AddBut	tons(UINT uNumButtons, LPTBBUTTON lpButtons) const;		
Adds one or mo	re buttons to the toolbar. AddString		
int AddStri	ng(UINT nStringID) const;		
Adds resource IDs to toolbar buttons. AddStrings			
int AddStri	ngs(LPCTSTR lpszStrings) const;		
Adds strings to	he toolbar's string pool. AutoSize		
void Autosi	ze() const;		

#### Resizes the toolbar. CheckButton

void CheckButton(int idButton, BOOL fCheck) const;

Checks or unchecks a given button in a toolbar. **CommandToIndex** 

int CommandToIndex(int idButton) const;

Returns the button index, given the command ID. Customize

void Customize() const;

Displays the Customize Toolbar dialog box. The parent must handle the TBN\_QUERYINSERT and TBN\_QUERYDELETE notifications for the Customize Toolbar dialog box to appear. **DeleteButton** 

B00L DeleteButton(int iButton) const;

Deletes a button from the toolbar DisableButton

B00L DisableButton(int idButton) const;

Disables a toolbar button. EnableButton

B00L EnableButton(int idButton) const;

Enables a toolbar button. GetButton

BOOL GetButton(int iButton, LPTBBUTTON lpButton) const;

Retrieves information about the specified button in the toolbar. GetButtonCount

int GetButtonCount() const;

Returns the number of buttons in the toolbar control. GetButtonSize

DWORD GetButtonSize() const;

Retrieves the current width and height of toolbar buttons, in pixels. GetButtonState

UINT GetButtonState(int idButton) const;

Returns the state of a toolbar button. GetButtonStyle

BYTE GetButtonStyle(int idButton) const

Returns the style of toolbar button. GetButtonText

LPCTSTR GetButtonText(int idButton) const;

Retrieves the display text of a button on the toolbar. GetCommandID

int GetCommandID(int iIndex) const;

Returns the command ID given the button index. GetDisabledImageList

HIMAGELIST GetDisabledImageList() const;

Retrieves the image list that the toolbar uses to display inactive buttons. GetHotImageList

HIMAGELIST GetImageList() const;

Retrieves the image list that the toolbar uses to display hot buttons. GetHotItem

int GetHotItem() const;

Retrieves the index of the hot button. GetImageList

HIMAGELIST GetImageList() const;

Retrieves the image list that the toolbar uses to display buttons in their default state. GetItemRect

CRect GetItemRect(int iIndex) const;

Returns the bounding rectangle of a button in a toolbar. GetMaxSize

CSize GetMaxSize() const;

Returns the SIZE required to contain the toolbar's buttons. GetPadding

DWORD GetPadding() const;

Retrieves the padding for the toolbar. GetRect

CRect GetRect(int idButton) const;

Retrieves the bounding rectangle for a specified toolbar button. GetRows

int GetRows() const;

Retrieves the number of rows of buttons in the toolbar. GetTextRows int GetTextRows() const;

Retrieves the maximum number of text rows that can be displayed on a toolbar button. GetToolBarData

std::vector<UINT>& GetToolBarData() const

Returns a reference to the vector of resource IDs for toolbar buttons. GetToolTips

CToolTip\* GetToolTips() const;

Retrieves the handle to the TooTip control, if any, associated with the toolbar. HasText

B00L HasText() const;

Returns TRUE if the button has text. HideButton

BOOL HideButton(int idButton, BOOL fShow) const;

Hides or shows the specified toolbar button. HitTest

int HitTest() const;

Determines which button a point lies in a toolbar control. Indeterminate

BOOL Indeterminate(int idButton, BOOL fIndeterminate) const;

Sets or clears the indeterminate state of the specified toolbar button. InsertButton

BOOL InsetButton(int iButton, LPTBBUTTON lpButton) const;

Inserts a button in the toolbar. IsButtonHidden

B00L IsButtonHidden(int idButton) const;

Determines whether the specified toolbar button is hidden. Is Button Highlighted

BOOL IsButtonHighlighted(int idButton) const;

Checks the highlight state of the toolbar button. IsButtonIndeterminate

B00L IsButtonIndeterminate(int idButton) const;

Determines whether the specified toolbar button is indeterminate. IsButtonPressed

B00L IsButtonPressed(int idButton) const;

Determines whether the specified toolbar button is pressed. MapAccelerator

int MapAccelerator(TCHAR chAccel) const;

Determines the ID of the button that corresponds to the specified accelerator character. MarkButton

B00L MarkButton(int idButton) const;

Sets the highlight state of a given button in a toolbar control. MoveButton

BOOL MoveButton(UINT uOldPos, UINT uNewPos) const;

Moves a button from one index to another. PressButton

BOOL PressButton(int idButton, BOOL fPress) const;

Presses or releases the specified toolbar button. ReplaceBitmap

virtual BOOL ReplaceBitmap(UINT NewToolBarID);

Replaces a toolbar's bitmap. See also SetImages. SaveRestore

void SaveRestore(BOOL fSave, TBSAVEPARAMS\* ptbsp) const;

Initiates saving or restoring the toolbar's state. SetBitmap

virtual BOOL SetBitmap(UINT nID);

Sets a toolbar's bitmap. See also SetImages. SetBitmapSize

B00L SetBitmapSize(int cx, int cy) const;

Sets the size of the bitmapped images to be added to a toolbar. Use this before adding the bitmap. SetButtonInfo

void SetButtonInfo(int idButton, int idButtonNew, int iImage, BYTE Style

Use this to change the buttons image and ID. The ID must be changed for the image to be changed. **SetButtonSize** 

BOOL SetBitmapSize(int cx, int cy) const;

Sets the size of the buttons to be added to a toolbar. SetButtonState

B00L SetButtonState(int idButton, UINT State) const;

Sets the state of a toolbar button. SetButtonStyle

BOOL SetButtonStyle(int idButton, BYTE Style) const;

Sets the style of a toolbar button. SetButtonText

virtual BOOL SetButtonText(int idButton, LPCTSTR szText);

Sets the text for a toolbar button. SetButtonWidth

B00L SetButtonWidth(int idButton, int nWidth) const;

Adjusts the width of a toolbar button after it is created. This is useful when replacing a button with a ComboBox or other control. **SetCommandID** 

BOOL SetCommandID(int iIndex, int idButton) const;

Sets the Command ID of a toolbar button. SetDisabledImageList

HIMAGELIST SetDisableImageList(HIMAGELIST himlNewDisabled) const;

Sets the image list that the toolbar will use to display disabled buttons. SetDrawTextFlags

DWORD SetDrawTextFlags(DWORD dwMask, DWORD dwDTFlags) const;

Sets the text drawing flags for the toolbar. SetExtendedStyle

DWORD SetExtendedStyle(DWORD dwExStyle) const;

Sets the extended styles for the toolbar. SetHotImageList

HIMAGELIST SetHotImageList(HIMAGELIST himlNewHot) const;

Sets the image list that the toolbar will use to display hot buttons. SetHotItem

int SetHotItem(int iHot);

Sets the hot item in the toolbar. This message is ignored for toolbars that do not have the TBSTYLE\_FLAT style. SetImageList

HIMAGELIST SetImageList(HIMAGELIST himlNew) const;

Sets the image list that the toolbar will use to display buttons that are in their default state. SetIndent

B00L SetIndent(int iIndent) const;

Sets the indentation for the first toolbar button. SetMaxTextRows

BOOL SetMaxTextRows(int iMaxRows) const;

Sets the maximum number of text rows displayed on a toolbar button. SetPadding

BOOL SetPadding(int cx, int cy) const;

Sets the padding for the toolbar. SetToolTips

```
void SetToolTips(CToolTip* pToolTip) const;
```

Associates a ToolTip control with a toolbar.

**Base class Members** 

For base class members, refer to the members of <u>CWnd</u>.

### Remarks

Like all common controls, the toolbar control requires a parent window. This parent window is often a <u>dialog</u>, but <u>simple windows</u> can also be the parent window for a toolbar control.

The following code demonstrates how to display a ComboBoxEx control in a toolbar. Before displaying the control over the 'File Save' button, the button's width is adjusted, and the button is converted to a separator.

```
void CMainFrame::AddCombo()
{
    // We'll be placing the ComboBoxEx control over the 'File Save' toolba
    int nComboWidth = 120;
    CToolBar& TB = GetToolBar();
    if (TB.CommandToIndex(IDM_FILE_SAVE) < 0) return;

    TB.SetButtonStyle(IDM_FILE_SAVE, TBSTYLE_SEP); // Convert the b
    TB.SetButtonWidth(IDM_FILE_SAVE, nComboWidth);

    // Determine the size and position of the ComboBox
    int nIndex = TB.CommandToIndex(IDM_FILE_SAVE);
    CRect rc = TB.GetItemRect(nIndex);

    // Create and position the ComboboxEx window
    m_ComboBoxEx.Create(TB.GetHwnd());
    m_ComboBoxEx.SetWindowPos(NULL, rc, SWP_NOACTIVATE);
</pre>
```

```
// Set ComboBox Height
m_ComboBoxEx.SendMessage(CB_SETITEMHEIGHT, (WPARAM)-1, (LPARAM)rc.Heig
m_ComboBoxEx.AddItems();
RecalcLayout();
}
```

Refer to the remarks section in the documentation for <u>CFrame</u> for an illustration on how to define an configure the toolbar.

### **Summary Information**

Header file	toolbar.h
Win32/64 support	Yes
WinCE support	Yes
Library required	Comctl32.lib

## **Tutorials**

### Menu of tutorials

Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance

- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing

Tutorial 10: Finishing Touches

#### **Tutorial 1: The Simplest Window**

The following code uses Win32++ to create a window. This is all the code you need (in combination with Win32++) to create and display a simple window. Note that in order to add the Win32++ code to our program, we use an #include statement as shown below.

```
#include "wincore.h"
```

```
// Note:
// * Add the Win32++\include directory to project's additional include
// A class that inherits from CWnd. It is used to create the window.
class CMyWindow : public CWnd
{
public:
    CMyWindow() {}
    virtual void OnDestroy() { PostQuitMessage(0); } // Ends the prog
    virtual ~CMyWindow() {}
};
int APIENTRY WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    // Start Win32++
    CWinApp MyApp;
    // Create a CMyWindow object
   CMyWindow MyWindow;
   // Create (and display) the window
    MyWindow.Create();
```

```
// Run the application's message loop
return MyApp.Run();
}
```

This program has four key steps:

- Start Win32++. We do this here by creating a CWinApp object called MyApp.
- Create a CMyWindow object called MyWindow.
- Create a default window by calling the Create function.
- Start the message loop, by calling the Run function.

The source code for this tutorial is located within the Tutorial folder of the software available from SourceForge at <u>http://sourceforge.net/projects/win32-framework</u>.

The CMyWindow class inherits from CWnd. CWnd is the base class for all objects used to create windows. We override the OnDestroy function of CWnd to end the program when the window is closed.

## **Tutorials**

#### Menu of tutorials

Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance

- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing

Tutorial 10: Finishing Touches

#### **Tutorial 2: Using Classes and Inheritance**

The program in the previous tutorial calls the CWinApp class directly. Normally, however, we would inherit from this class to have more control over the type of CWinApp objects we create.

This is an example of how we would derive a class from CWinApp.

```
// A class that inherits from CWinApp.
// It is used to run the application's message loop.
class CSimpleApp : public CWinApp
{
public:
    CSimpleApp() {}
    virtual ~CSimpleApp() {}
    virtual BOOL InitInstance();
private:
    CView m View;
};
BOOL CSimpleApp::InitInstance()
{
    // Create the Window
    m View.Create();
    return TRUE;
```

}

Notice that we override InitInstance to determine what happens when the application is started. In this instance we create the window for the m\_View member variable. The m\_View member variable is a CView object inherited from CWnd. The code for CView is shown below.

// \* Add the Win32++\include directory to project's additional include

```
#include "wincore.h"
// A class that inherits from CWnd. It is used to create the window.
class CView : public CWnd
{
public:
    CView() {}
    virtual void OnDestroy() { PostQuitMessage(0); } // Ends the program
    virtual ~CView() {}
};
```

The CSimpleApp and CView classes are used in WinMain as follows.

```
int APIENTRY WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    // Start Win32++
    CSimpleApp MyApp;
    // Run the application
    return MyApp.Run();
}
```

The source code for this tutorial is located within the Tutorial folder of the software available from SourceForge at <u>http://sourceforge.net/projects/win32-framework</u>.

## **Tutorials**

#### Menu of tutorials

- Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance
- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing

Tutorial 10: Finishing Touches

#### **Tutorial 3: Using Messages to create a Scribble Application**

Each external event that a window might need to respond to is sent to the window by way of a message. Its now time to control the way our window behaves by handling some of these messages. Each **CWnd** object handles its own messages in the **WndProc** function. In this example we will create a simple scribble program by handling the left mouse button messages. We allow the user to draw on the window by responding to the mouse messages. A line is drawn on the window when the user moves the mouse while pressing the left mouse button down.

The WM\_LBUTTONDOWN message is sent to the window when the left button is clicked, and the cursor is over the window's client area. We capture the mouse input and store the current mouse position in the m\_OldPt member variable.

The WM\_MOUSEMOVE message is posted to a window when the cursor moves. If the mouse is not captured, the message is posted to the window that contains the cursor. Otherwise, the message is posted to the window that has captured the mouse. We check that the left button is also down, and call DrawLine to draw the line in the view window.

The WM\_LBUTTTONUP message is sent to the window when the left mouse button changed from down to up during mouse capture.

A window receives messages through its window procedure. We intercept these messages and take our own actions by overriding the WndProc function.

```
LRESULT CView::WndProc(UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
        case WM_LBUTTONDOWN:
            OnLButtonDown(lParam);
            break;
        case WM_MOUSEMOVE:
```

```
OnMouseMove(wParam, lParam);
break;
case WM_LBUTTONUP:
    OnLButtonUp(lParam);
    break;
}
//Use the default message handling for remaining messages
return WndProcDefault(uMsg, wParam, lParam);
}
```

These are the definitions of the functions used in WndProc. When the WM\_LBUTTONDOWN message is received, we use SetCapture to capture the mouse input. This allows our window to receive the mouse messages even if the mouse cursor is moved outside our window. We stop capturing the mouse input with ReleaseCapture when the left mouse button is released.

```
void CView::OnLButtonDown(LPARAM lParam)
{
  // Capture mouse input.
  SetCapture();
 m OldPt.x = GET X LPARAM(lParam);
  m OldPt.y = GET Y LPARAM(lParam);
}
void CView::OnLButtonUp(LPARAM lParam)
{
  {
    //Release the capture on the mouse
    ReleaseCapture();
  }
}
void CView::OnMouseMove(WPARAM wParam, LPARAM lParam)
{
```

```
// hold down the left mouse button and move mouse to draw lines.
if (wParam & MK_LBUTTON)
{
    DrawLine(LOWORD(lParam), HIWORD(lParam));
    m_OldPt.x = GET_X_LPARAM(lParam);
    m_OldPt.y = GET_Y_LPARAM(lParam);
}
```

# Menu of tutorials

- Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance
- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing
- Tutorial 10: Finishing Touches

# **Tutorial 4: Repainting the Window**

The previous example fails to repain the window. Repainting the window is required whenever the window is resized, restored from minimized, or when part of the window is revealed after being covered by another window.

The Windows API handles repainting automatically. When part of a window needs repainting, windows sends a WM\_PAINT message to the application. Typically you would respond to this message with the code to redraw the entire client area of the window. You don't need to concern yourself with which parts of the client area need to be redrawn, as windows handles this part for you automatically.

Win32++ already contains the code to handle the WM\_PAINT message in CWnd::WndProc. All we need to do is write the OnDraw function. For our application here, we can store the various points in a vector, and have the OnDraw function draw the lines again. The function to store the points looks like this.

```
void CView::StorePoint(int x, int y, bool PenDown)
{
    PlotPoint P1;
    Pl.x = x;
    Pl.y = y;
    Pl.PenDown = PenDown;
    m_points.push_back(P1); //Add the point to the vector
}
```

Our OnDraw function looks like this.

```
void CView::OnDraw(CDC* pDC)
{
    if (m_points.size() > 0)
    {
        bool bDraw = false; //Start with the pen up
```

```
for (unsigned int i = 0 ; i < m_points.size(); i++)
{
    if (bDraw)
        pDC->LineTo(m_points[i].x, m_points[i].y);
    else
        pDC->MoveTo(m_points[i].x, m_points[i].y);
    bDraw = m_points[i].PenDown;
    }
}
```

#### Menu of tutorials

- Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance
- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing

Tutorial 10: Finishing Touches

case IDM FILE NEW:

#### **Tutorial 5: Wrapping a Frame around our Scribble Application**

This is the code we use to produce a simple frame application. We inherit a class called CMainframe from the CFrame class in Win32++. Notice how we use the SetView function to specify the "view" window of our frame. This "view" window happens to be the same code as the simple scribble application shown in the previous tutorial. In this way, we wrap a frame around our previous scribble application.

The CMainFrame class inherits OnCommand from CFrame. This function responds to input from the frame's menu and toolbar. We haven't implemented many of these inputs yet, so at this stage most of them simply display a simple message box.

```
CMainFrame::CMainFrame()
{
  //Set m View as the view window of the frame
  SetView(m View);
}
BOOL CMainFrame::OnCommand(WPARAM wParam, LPARAM lParam)
{
  // Process the messages from the Menu and Tool Bar
  switch (LOWORD(wParam))
  {
  case IDM FILE EXIT:
    // End the application
    ::PostMessage(m hWnd, WM CLOSE, 0, 0);
    return TRUE:
  case IDM HELP ABOUT:
    // Display the help dialog
    OnHelp();
    return TRUE;
```

::MessageBox(NULL, \_T("File New ... Implemented later"), \_T("Menu")

```
return TRUE;
 case IDM FILE OPEN:
    ::MessageBox(NULL, _T("File Open ... Implemented later"), T("Menu"
    return TRUE;
 case IDM FILE SAVE:
    ::MessageBox(NULL, T("File Save ... Implemented later"), T("Menu"
    return TRUE;
 case IDM_FILE_SAVEAS:
    ::MessageBox(NULL, _T("File SaveAs ... Implemented later"), T("Men
    return TRUE;
 case IDM_FILE PRINT:
    ::MessageBox(NULL, _T("File Print ... Implemented later"), _T("Menu
    return TRUE;
 }
 return FALSE;
}
```

# Menu of tutorials

Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance

- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing

Tutorial 10: Finishing Touches

# **Tutorial 6: Customising Window Creation**

Up until now we have used the default parameters supplied by Win32++ to create the view window. Here we use the PreRegisterClass to specify the Window Class (not to be confused with a C++ class) parameters prior to window creation. This will allow us to create a window with a coloured background and set it's cursor.

```
void CView::PreRegisterClass(WNDCLASS &wc)
{
    // Set the background brush and cursor
    wc.hbrBackground = m_Brush;
    wc.lpszClassName = "Scribble Window";
    wc.hCursor = GetApp()->LoadCursor(IDC_CURSOR1);
}
```

We also use PreCreate to give the window a 3D look by giving it's border a sunken edge.

```
void CView::PreCreate(CREATESTRUCT &cs)
{
   // Set the extra style to provide a sunken edge
   cs.dwExStyle = WS_EX_CLIENTEDGE;
}
```

The cursor IDC\_CURSOR1 used in PreRegisterClass is specified in resource.rc. The resource.rc file is our resource script and contains the specifications for various window resources such as bitmaps, dialogs, cursors, icons, menus etc. The resources specified in resource.rc are compiled and linked into our application. This is the specification for the cursor in the resource.rc file.

IDC_CURSOR1	CURSOR	"res/Cursor.cur"
-------------	--------	------------------

We create the background brush in CView's constructor. The brush is a CBrush object which automatically deletes its brush when its destructor is called. This is how the brush is created.

# Menu of tutorials

- Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance
- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing

Tutorial 10: Finishing Touches

#### **Tutorial 7: Customising the ToolBar**

Our frame window has both a menu and a toolbar at the top. Customising the menu is relatively straight forward, since we can use a resource editor to perform that task. Customising the toolbar however is another matter. While its true that the resource editor which ships Microsoft's Visual Studio can edit the toolbar resource, this is not standard, and we so need to come up with a standards compliant way of modifying the toolbar.

To set your own toolbar for your applications, you will need to perform the following steps:

- Modify the toolbar bitmap.
- Add the toolbar resource IDs to to the string table.
- Assign resource IDs to the toolbar buttons, as shown below.

```
void CMainFrame::SetupToolBar()
{
  // Define the resource IDs for the toolbar
 AddToolBarButton( IDM FILE NEW
                                    );
  AddToolBarButton( IDM FILE OPEN
                                    );
  AddToolBarButton( IDM FILE SAVE
                                    );
  AddToolBarButton( 0 );
                                                          // Separator
 AddToolBarButton( IDM EDIT CUT,
                                     FALSE ):
 AddToolBarButton( IDM_EDIT_COPY,
                                     FALSE );
 AddToolBarButton( IDM EDIT PASTE, FALSE );
 AddToolBarButton( 0 );
                                                          // Separator
 AddToolBarButton( IDM FILE PRINT );
 AddToolBarButton( 0 );
                                                          // Separator
  AddToolBarButton( IDM HELP ABOUT );
}
```

# Menu of tutorials

- Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance
- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing
- Tutorial 10: Finishing Touches

# **Tutorial 8: Loading and Saving Files**

In this tutorial we will demonstrate how to save our data to a file, and load it back again.

# Saving Data

To save the data we perform the following steps:

- Use CFile::SaveFileDialog to open a dialog and choose the filename to save the data to.
- Use CFile::Open to open the file for writing.
- Use CFile::Write to write the data to the file.

This code segment shows how to use the SaveFileDialog function to choose the file to write to.

```
void CMainFrame::OnFileSaveAs()
{
    CFile File;
    CString str =
    File.SaveFileDialog(0, OFN_OVERWRITEPROMPT, _T("Save File"), _T("Scrib
    // Store the PlotPoint data in the file
    if (!str.IsEmpty())
    {
        m_PathName = str;
        // Save the file name
        m_View.FileSave(str);
        AddMRUEntry(str);
    }
}
```

This next code segment demonstrates how to open a file for writing, and write data to it. The data written in this case is the contents of the m\_points vector. Error checking has been omitted from the code segment displayed here for clarity, but is included in the code for this tutorial.

```
B00L CView::FileSave(LPCTSTR szFilename)
{
    B00L bResult = TRUE;
    CFile hFile;
    hFile.Open(szFilename, CREATE_ALWAYS))
    // Write the file
for
    (size_t i =
        0; i < m_points.size(); ++i)
    {
        hFile.Write(&m_points[i], sizeof(PlotPoint));
    }
    return bResult;
}</pre>
```

# **Loading Data**

Reading data back from a file follows a rather similar process. The steps involved are as follows:

- Use CFile::OpenFileDialog to open a dialog and choose the file to load.
- Use CFile::Open to open the file for reading.
- Use CFile::Read to retrieve the data from the file.

The following code segment demonstrates how to use OpenFileDialog to choose the file to load.

```
void CMainFrame::OnFileOpen()
{
    CFile File;
    CString str =
    File.OpenFileDialog(0, OFN_FILEMUSTEXIST, _T("Open File"), _T("Scribbl
    if (!str.IsEmpty())
    {
        // Retrieve the PlotPoint data
        if (m_View.FileOpen(str))
        {
            // Save the filename
            m_PathName = str;
            AddMRUEntry(str);
        }
    }
}
```

```
}
else
m_PathName=_T("");
}
```

This is the code which loads our PlotPoint vector from the contents of the chosen file. Once again, the error checking has been omitted from the code displayed here for clarity, but is present in the source code.

```
BOOL CView::FileOpen(LPCTSTR szFilename)
{
  // empty the PlotPoint vector
 m points.clear();
 DWORD nBytesRead;
  BOOL bResult = FALSE;
  // Create a handle to the file
 CFile File;
 if (File.Open(szFilename, OPEN EXISTING))
  {
    do
    {
      nBytesRead = 0;
      PlotPoint pp;
      nBytesRead = File.Read(&pp, sizeof(PlotPoint));
      if (nBytesRead == sizeof(PlotPoint))
        m_points.push_back(pp);
    } while (nBytesRead == sizeof(PlotPoint));
    bResult = TRUE;
 }
 Invalidate();
  return bResult;
}
```

#### Menu of tutorials

- Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance
- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files

Tutorial 9: Printing

Tutorial 10: Finishing Touches

#### **Tutorial 9: Printing**

In this tutorial we will demonstrate how to send a bitmap to a printer. The bitmap in this case will be the one we've been drawing on in the view window. The resulting printout will be resized to match the size of the original drawing. This task can be broken down into several steps:

- Extract the bitmap from the view window
- Choose the printer
- Start the print job
- Extract the bitmap image data from the bitmap
- Copy the resized image data to the printer's device context
- End the print job

#### Extract the bitmap from the view window

When we draw on the view window, we are actually drawing on to a bitmap attached to view window's device context. Here we will copy this bitmap to a compatible bitmap.

// Copy the bitmap from the View window CClientDC dcView(&m;\_View); CMemDC MemDC(&dcView;); CBitmap bmView; bmView.CreateCompatibleBitmap(&dcView, Width, Height); MemDC.SelectObject(&bmView;); MemDC.BitBlt(0, 0, Width, Height, &dcView;, 0, 0, SRCCOPY);

# **Choose a printer**

This step is fairly straight forward. We declare a PRINTDLG struct and use this in the PrintDlg function. The PrintDlg function brings up a dialog, allowing us to choose the printer, and stores its device context in the PRINTDLG struct.

// Bring up a dialog to choose the printer
PRINTDLG pd = {0};

```
pd.lStructSize = sizeof(pd);
pd.Flags = PD_RETURNDC;
pd.hwndOwner = m_hWnd;
// Retrieve the printer DC
PrintDlg(&pd);
```

#### **Start the Print Job**

The StartDoc function should be called before sending output to a printer. This function ensures that multipage documents are not interspersed with other print jobs. StartPage (and a corresponding EndPage) is then called for each page of printout.

```
// Zero and then initialize the members of a DOCINFO structure.
DOCINFO di = {0};
di.cbSize = sizeof(DOCINFO);
di.lpszDocName = _T("Scribble Printout");
di.lpszOutput = (LPTSTR) NULL;
di.lpszDatatype = (LPTSTR) NULL;
di.fwType = 0;
// Begin a print job by calling the StartDoc function.
StartDoc(pd.hDC, &di);
// Inform the driver that the application is about to begin sending data
StartPage(pd.hDC);
```

#### Extract the bitmap image data from the bitmap

In order to use StretchDIBits we first need the bitmap image data. This is retrieved by using the GetDIBits. It is called twice in the following code sample, the first time to get the size byte array to hold the data, and the second to fill the byte array.

```
// Get the dimensions of the View window
RECT rcView:
GetClientRect (m View.GetHwnd(), &rcView);
int Width = rcView.right - rcView.left;
int Height = rcView.bottom - rcView.top;
// Fill the BITMAPINFOHEADER structure
BITMAPINFOHEADER bi = \{0\};
bi.biSize = sizeof(BITMAPINFOHEADER);
bi.biHeight = Height;
bi.biWidth = Width;
bi.biPlanes = 1;
bi.biBitCount = 24;
bi.biCompression = BI_RGB;
// Note: BITMAPINFO and BITMAPINFOHEADER are the same for 24 bit bitmaps
// Get the size of the image data
MemDC.GetDIBits(&bmView, 0, Height, NULL, (BITMAPINF0*)&bi, DIB RGB COLO
```

```
// Retrieve the image data
byte* pBits = new byte[bi.biSizeImage];
MemDC.GetDIBits(&bmView, 0, Height, pBits, (BITMAPINFO*)&bi, DIB_RGB_COL
```

### Copy the resized bitmap to the printer's device context

StretchDIBits is the function used here to copy the bitmap information to the printer's device context because the bitmap needs to be resized in order to retain the same dimensions on the printed page as the original. The following code segment shows how the scaling factors are calculated and the StretchDIBits function is called.

```
// Determine the scaling factors required retain the bitmap's original p
float fLoqPelsX1 = (float) GetDeviceCaps(ViewDC, LOGPIXELSX);
float fLogPelsY1 = (float) GetDeviceCaps(ViewDC, LOGPIXELSY);
float fLogPelsX2 = (float) GetDeviceCaps(pd.hDC, LOGPIXELSX);
float fLogPelsY2 = (float) GetDeviceCaps(pd.hDC, LOGPIXELSY);
float fScaleX = MAX(fLogPelsX1, fLogPelsX2) / min(fLogPelsX1, fLogPelsX2
float fScaleY = MAX(fLogPelsY1, fLogPelsY2) / min(fLogPelsY1, fLogPelsY2
// Compute the coordinates of the upper left corner of the centered bitm
int cWidthPels = GetDeviceCaps(pd.hDC, HORZRES);
int xLeft = ((cWidthPels / 2) - ((int) (((float) Width) * fScaleX)) / 2)
int cHeightPels = GetDeviceCaps(pd.hDC, VERTRES);
int yTop = ((cHeightPels / 2) - ((int) (((float) Height) * fScaleY)) / 2
// Use StretchDIBits to scale the bitmap and maintain its original propo
StretchDIBits(pd.hDC, xLeft, yTop, (int) ((float) Width * fScaleX),
               (int) ((float) Height * fScaleY), 0, 0, Width, Height, pB
               (BITMAPINFO*)&bi, DIB RGB COLORS, SRCCOPY);
```

#### End the print job

To finish the print job, EndPage is called to indicate that printing to this page is complete, then EndDoc is called the end the print job.

// Inform the driver that the page is finished. EndPage(pd.hDC);

// Inform the driver that document has ended. EndDoc(pd.hDC);

# Menu of tutorials

- Tutorial 1: The Simplest WindowTutorial 2: Using Classes and Inheritance
- Tutorial 3: Using Messages to Create a Scribble Window
- Tutorial 4: Repainting the Window
- Tutorial 5: Wrapping a Frame around our Scribble Window
- Tutorial 6: Customising Window Creation
- Tutorial 7: Customising the ToolBar
- Tutorial 8: Loading and Saving Files
- Tutorial 9: Printing
- Tutorial 10: Finishing Touches

# **Tutorial 10: Finishing Touches**

In this final tutorial we will show how to store the application's settings in the registry, and implement tracing. The code for this final tutorial is the scribble demo application.

#### Saving the Window Position

Users will expect modern applications to save their settings, such as the position and size of the frame window. These settings are stored in the registry. LoadRegistrySettings is used to set the name of the registry key. Typically the name takes the form of "CompanyName\\ApplicationName" as demonstrated below.

```
CMainFrame::CMainFrame()
{
    .
    .
    .
    .
    // Set the registry key name, and load the initial window position
    // Use a registry key name like "CompanyName\\Application"
    LoadRegistrySettings(_T("Win32++\\Scribble Sample"));
}
```

If a registry key name has been set using the LoadRegistrySetting function, the registry settings will be loaded from the registry when the application starts, and stored in the registry when the application ends. Override the LoadRegistrySettings and SaveRegistrySettings functions in CMainFrame if you wish to store other settings in the registry.

# Saving the Most Recently Used (MRU) List

Applications which load and store files typically allow the user to choose from a list of recently used file names to load from. This MRU list is also stored in the registry. To add this capability to your Win32++

application, use the LoadRegistryMRUSettings function to specify the number of entries in the MRU list (up to a maximum of 16) as shown below.

```
CMainFrame::CMainFrame()
```

```
{
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
    .
```

To see the MRU entries listed in the menu, add "Recent Files" menu item to the menu definitions in Resource.rc. as follows:

MENUITEM "Recent Files", IDW FILE MRU FILE1, GRAYED

MRU entries are added with AddMRUEntry, and removed with RemoveMRUEntry.

# **Command Line Arguments**

Command line arguments are passed to the program when it is started. The GetCommandLine function is used to retrieve the the command line string. The CommandLineToArgvW can parse this string and convert it to an array of LPWSTR pointers.

The following code demonstrates how to load a file when the filename is supplied as a command line argument.

```
void CMainFrame::OnInitialUpdate()
{
    int argCount = 0;
    LPWSTR* lpArgv = ::CommandLineToArgvW(::GetCommandLineW(), &argCount;)
    // The second argument (if any) contains our file name.
    if (argCount >= 2)
    {
        m_View.FileOpen((W2T(lpArgv[1])));
    }
}
```

This allows the application to start and load the file when it is selected and opened from within Windows Explorer.

#### **Debugging your Application with Tracing**

One important debugging technique is to trace what is happening with the application while it is running.

The tracing allows you to display messages or the contents of variables in the output pane of Visual Studio (or whichever Integrated Development Environment you use).

In order to take advantage of this, you will need do the following

- Use Version 6.0 (or later) of Win32++
- Run your application in debug mode (see below)
- Use the TRACE function to display the messages.

To run the application in debug mode, you need to have the \_DEBUG variable defined. Microsoft's Visual Studio products define this variable for you when you compile in debug mode. For other compilers you should use the appropriate compiler option to set this variable. The source code for this example, as well as the samples in the download section, include project files for debug modes for both DevC++ and CodeBlocks to make this task easier.

If you are not running in debug mode, TRACE statements have no effect. You can leave them in place for Release mode if you wish.

In the sample code for this section we add tracing to our scribble application to display the position of the mouse for the lines we draw.

This is the code added to CView::WndProc for tracing.

These are the lines of code added to CMainFrame::WndProc

```
TRACE("Red pen selected\n");
TRACE("Blue pen selected\n");
```

```
TRACE("Green pen selected\n");
```

```
TRACE("Black pen selected\n");
```