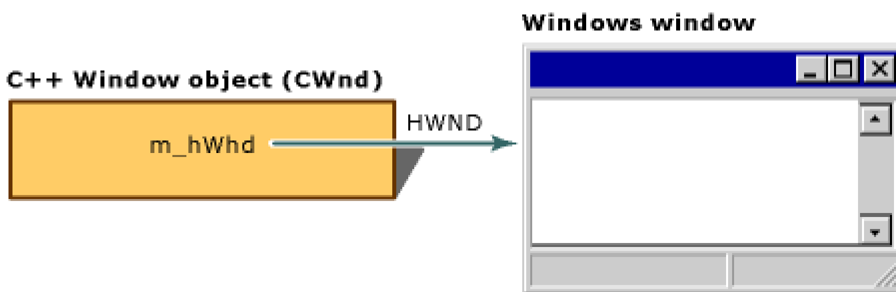
Relationship Between a C++ Window Object and an HWND

The window *object* is an object of the C++ `CWnd` class (or a derived class) that your program creates directly. It comes and goes in response to your program's constructor and destructor calls. The Windows *window*, on the other hand, is an opaque handle to an internal Windows data structure that corresponds to a window and consumes system resources when present. A Windows window is identified by a "window handle" (`HWND`) and is created after the `CWnd` object is created by a call to the **Create** member function of class `CWnd`. The window may be destroyed either by a program call or by a user's action. The window handle is stored in the window object's `m_hWnd` member variable. The following figure shows the relationship between the C++ window object and the Windows window. Creating windows is discussed in Creating Windows. Destroying windows is discussed in Destroying Window Objects.



Window Object and Windows Window

Working with Window Objects

Working with windows calls for two kinds of activity:

- Handling Windows messages
- Drawing in the window

To handle Windows messages in any window, including your own child windows, see Mapping Messages to Functions to map the messages to your C++ window class. Then write message-handler member functions in your class.

Most drawing in a framework application occurs in the view, whose OnDraw member function is called whenever the window's contents must be drawn. If your window is a child of the view, you might delegate some of the view's drawing to your child window by having `OnDraw` call one of your window's member functions.

In any case, you will need a device context for drawing. You can use the stock pen, brush, and other graphic objects contained in the device context associated with your window. Or you can modify these objects to get the drawing effects you need. With your device context set up as you like, call member functions of class CDC (device-context class) to draw lines, shapes, and text; to use colors; and to work with a coordinate system.

Message Handling and Mapping

This article family describes how messages and commands are processed by the MFC framework and how you connect them to their handler functions.

In traditional programs for Windows, Windows messages are handled in a large switch statement in a window procedure. MFC instead uses message maps to map direct messages to distinct class member functions. Message maps are more efficient than virtual functions for this purpose, and they allow messages to be handled by the most appropriate C++ object — application, document, view, and so on. You can map a single message or a range of messages, command IDs, or control IDs.

WM_COMMAND messages — usually generated by menus, toolbar buttons, or accelerators — also use the message-map mechanism. MFC defines a standard routingof command messages among the application, frame window, view, and Active documents in your program. You can override this routing if you need to.

Message maps also supply a way to update user-interface objects (such as menus and toolbar buttons), enabling or disabling them to suit the current context.

For general information about messages and message queues in Windows, see Messages and Message Queues in the Windows SDK.

Device Contexts

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 device-independent drawing in Windows. Device contexts can be used to draw to the screen, to the printer, or to a metafile.

CPaintDC objects encapsulate the common idiom of Windows, calling the `BeginPaint`function, then drawing in the device context, then calling the `EndPaint` function.
The `CPaintDC` constructor calls `BeginPaint` for you, and the destructor calls `EndPaint`. The simplified process is to create the CDC object, draw, and then destroy the `CDC` object. In the framework, much of even this process is automated. In particular, your `OnDraw`function is passed a `CPaintDC` already prepared (via `OnPrepareDC`), and you simply draw into it. It is destroyed by the framework and the underlying device context is released to Windows upon return from the call to your `OnDraw` function.

CClientDC objects encapsulate working with a device context that represents only the client area of a window. The `CClientDC` constructor calls the `GetDC` function, and the destructor calls the `ReleaseDC` function. CWindowDC objects encapsulate a device context that represents the whole window, including its frame.

CMetaFileDC objects encapsulate drawing into a Windows metafile. In contrast to the `CPaintDC` passed to `OnDraw`, you must in this case call OnPrepareDC yourself.
Mouse Drawing

Most drawing in a framework program — and thus most device-context work — is done in the view's `OnDraw` member function. However, you can still use device-context objects for other purposes. For example, to provide tracking feedback for mouse movement in a view, you need to draw directly into the view without waiting for `OnDraw` to be called.

In such a case, you can use a CClientDC device-context object to draw directly into the view.

Window Objects

In this article

1. Functions for Operating On a CWnd
2. CWnd and Windows Messages
3. See Also

The latest version of this topic can be found at Window Objects.

MFC supplies class CWnd to encapsulate the `HWND` handle of a window. The `CWnd`object is a C++ window object, distinct from the `HWND` that represents a Windows window but containing it. Use `CWnd` to derive your own child window classes, or use one of the many MFC classes derived from `CWnd`. Class `CWnd` is the base class for all windows, including frame windows, dialog boxes, child windows, controls, and control bars such as toolbars. A good understanding of the relationship between a C++ window object and an HWND is crucial for effective programming with MFC.

MFC provides some default functionality and management of windows, but you can derive your own class from `CWnd` and use its member functions to customize the provided functionality. You can create child windows by constructing a `CWnd` object and calling its Create member function, then customize the child windows using `CWnd`member functions. You can embed objects derived from CView, such as form views or tree views, in a frame window. And you can support multiple views of your documents via splitter panes, supplied by class CSplitterWnd.

Each object derived from class `CWnd` contains a message map, through which you can map Windows messages or command IDs to your own handlers.

The general literature on programming for Windows is a good resource for learning how to use the `CWnd` member functions, which encapsulate the `HWND` APIs.

Functions for Operating On a CWnd

`CWnd` and its derived window classes provide constructors, destructors, and member functions to initialize the object, create the underlying Windows structures, and access the encapsulated `HWND`. `CWnd` also provides member functions that encapsulate Windows APIs for sending messages, accessing the window's state, converting coordinates, updating, scrolling, accessing the Clipboard, and many other tasks. Most Windows window-management APIs that take an `HWND` argument are encapsulated as member functions of `CWnd`. The names of the functions and their parameters are preserved in the `CWnd` member function. For details about the Windows APIs encapsulated by `CWnd`, see class CWnd.

CWnd and Windows Messages

One of the primary purposes of `CWnd` is to provide an interface for handling Windows messages, such as `WM_PAINT` or `WM_MOUSEMOVE`. Many of the member functions of `CWnd`are handlers for standard messages — those beginning with the identifier **afx_msg** and the prefix "On," such

as `OnPaint` and **OnMouseMove**. Message Handling and Mappingcovers messages and message handling in detail. The information there applies equally to the framework's windows and those that you create yourself for special purposes.

Detaching a CWnd from Its HWND

The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.

The latest version of this topic can be found at Detaching a CWnd from Its HWND.

If you need to circumvent the object-`HWND` relationship, MFC provides another `CWnd`member function, Detach, which disconnects the C++ window object from the Windows window. This prevents the destructor from destroying the Windows window when the object is destroyed.

Using the Classes to Write Applications for Windows

The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.

The latest version of this topic can be found at Using the Classes to Write Applications for Windows.

Taken together, the classes in the Microsoft Foundation Class (MFC) Library make up an "application framework," on which you build an application for the Windows operating system. At a very general level, the framework defines the skeleton of an application and supplies standard user-interface implementations that can be placed onto the skeleton. Your job as programmer is to fill in the rest of the skeleton, which are those things that are specific to your application. You can get a head start by using the MFC Application Wizard to create the files for a very thorough starter application. You use the Microsoft Visual C++ resource editors to design your user-interface elements visually, Class View commands to connect those elements to code, and the class library to implement your application-specific logic.

Version 3.0 and later of the MFC framework supports programming for Win32 platforms, including Microsoft Windows 95 and later, and Windows NT versions 3.51 and later. MFC Win32 support includes multithreading. Use version 1.5*x* if you need to do 16-bit programming.

This family of articles presents a broad overview of the application framework. It also explores the major objects that make up your application and how they are created. Among the topics covered in these articles are the following:

- The framework.
- Division of labor between the framework and your code, as described in Building on the Framework.
- The application class, which encapsulates application-level functionality.
- How document templates create and manage documents and their associated views and frame windows.
- Class CWnd, the root base class of all windows.
- Graphic objects, such as pens and brushes.

Other parts of the framework include:

- Window Objects: Overview
- Message handling and mapping
- CObject, The Root Base Class in MFC
- Document/View Architecture
- Dialog Boxes
- Controls
- Control Bars
- OLE
- Memory Management

  Besides giving you an advantage in writing applications for the Windows operating system, MFC also makes it much easier to write applications that specifically use OLE linking and embedding technology. You can make your application an OLE visual editing container, an OLE visual editing server, or both, and you can add Automation so that other applications can use objects from your application or even drive it remotely.

- MFC ActiveX Controls

  The OLE control development kit (CDK) is now fully integrated with the framework. This article family supplies an overview of ActiveX control development with MFC. (ActiveX controls were formerly known as OLE controls.)

- Database Programming

  MFC also supplies two sets of database classes that simplify writing data-access applications. Using the ODBC database classes, you can connect to databases through an Open Database Connectivity (ODBC) driver, select records from tables, and display record information in an on-screen form. Using the Data Access Object (DAO) classes, you can work with databases through the Microsoft Jet database engine or external (non-Jet) data sources, including ODBC data sources.

  In addition, MFC is fully enabled for writing applications that use Unicode and multibyte character sets (MBCS), specifically double-byte character sets (DBCS).

For a general guide to MFC documentation, see General MFC Topics.

See Also

General MFC Topics

Controls (MFC)

1. Windows Common Controls
2. ActiveX Controls
3. Other MFC Control Classes
4. Finding Information About Windows Common Controls
5. See Also

The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.

The latest version of this topic can be found at Controls (MFC).

Controls are objects that users can interact with to enter or manipulate data. They commonly appear in dialog boxes or on toolbars. This topic family covers three main kinds of controls:

- Windows common controls, including owner-drawn controls
- ActiveX Controls
- Other control classes supplied by the Microsoft Foundation Class Library (MFC)

Windows Common Controls

The Windows operating system has always provided a number of Windows common controls. These control objects are programmable, and the Visual C++ dialog editor supports adding them to your dialog boxes. The Microsoft Foundation Class Library (MFC) supplies classes that encapsulate each of these controls, as shown in the table Windows Common Controls and MFC Classes. (Some items in the table have related topics that describe them further. For controls that lack topics, see the documentation for the MFC class.)

Class CWnd is the base class of all window classes, including all of the control classes. The Windows common controls are supported in the following environments:

- Windows 95, Windows 98, and Windows 2000
- Windows NT, version 3.51 and later
- Win32s, version 1.3 (Visual C++ versions 4.2 and later do not support Win32s)

The older common controls — check boxes, combo boxes, edit boxes, list boxes, option buttons, pushbuttons, scroll bar controls, and static controls — were available in earlier versions of Windows as well.

ActiveX Controls

ActiveX controls, formerly known as OLE controls, can be used in dialog boxes in your applications for Windows, or in HTML pages on the World Wide Web. For more information, see MFC ActiveX Controls.

Other MFC Control Classes

In addition to classes that encapsulate all of the Windows common controls and that support programming your own ActiveX controls (or using ActiveX controls supplied by others), MFC supplies the following control classes of its own:

- CBitmapButton
- CCheckListBox
- CDragListBox

Finding Information About Windows Common Controls

The table below briefly describes each of the Windows common controls, including the control's MFC wrapper class.

Windows Common Controls and MFC Classes

Control	MFC class	Description	New in Windows 95
animation	CAnimateCtrl	Displays successive frames of an AVI video clip	Yes
button	CButton	Pushbuttons that cause an action; also used for check boxes, radio buttons, and group boxes	No
combo box	CComboBox	Combination of an edit box and a list box	No
date and time picker	CDateTimeCtrl	Allows the user to choose a specific date or time value	Yes
edit box	CEdit	Boxes for entering text	No
extended combo box	CComboBoxEx	A combo box control with the ability to display images	Yes
header	CHeaderCtrl	Button that appears above a column of text; controls width of text displayed	Yes
hotkey	CHotKeyCtrl	Window that enables user to create a "hot key" to perform an action quickly	Yes
image list	CImageList	Collection of images used to manage large sets of icons or bitmaps (image list isn't really a control; it supports lists used by other controls)	Yes
list	CListCtrl	Window that displays a list of text with icons	Yes
list box	CListBox	Box that contains a list of strings	No

Control	MFC class	Description	New in Windows 95
month calendar	CMonthCalCtrl	Control that displays date information	Yes
progress	CProgressCtrl	Window that indicates progress of a long operation	Yes
rebar	CRebarCtrl	Tool bar that can contain additional child windows in the form of controls	Yes
rich edit	CRichEditCtrl	Window in which user can edit with character and paragraph formatting (see Classes Related to Rich Edit Controls)	Yes
scroll bar	CScrollBar	Scroll bar used as a control inside a dialog box (not on a window)	No
slider	CSliderCtrl	Window containing a slider control with optional tick marks	Yes
spin button	CSpinButtonCtrl	Pair of arrow buttons user can click to increment or decrement a value	Yes
static-text	CStatic	Text for labeling other controls	No
status bar	CStatusBarCtrl	Window for displaying status information, similar to MFC class `CStatusBar`	Yes
tab	CTabCtrl	Analogous to the dividers in a notebook; used in "tab dialog boxes" or property sheets	Yes
toolbar	CToolBarCtrl	Window with command-generating buttons, similar to MFC class `CToolBar`	Yes
tool tip	CToolTipCtrl	Small pop-up window that describes purpose of a toolbar button or other tool	Yes

Control	MFC class	Description	New in Windows 95
tree	CTreeCtrl	Window that displays a hierarchical list of items	Yes

What do you want to know more about

- An individual control: see the table Windows Common Controls and MFC Classes in this topic for links to all controls
- Making and using controls
- Using the dialog editor to add controls
- Adding controls to a dialog box by hand
- Deriving control classes from the MFC control classes
- Using common controls as child windows
- Notifications from common controls
- Add common controls to a dialog box.
- Derive a control from a standard Windows control
- Access dialog-box controls with type safety
- Receive notification messages from common controls
- Samples

For information about Windows common controls in the Windows SDK, see Common Controls.

See Also

User Interface Elements
Dialog Editor

Dialog Boxes

1. What do you want to know more about
2. See Also

The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.

The latest version of this topic can be found at Dialog Boxes.

Applications for Windows frequently communicate with the user through dialog boxes. Class CDialog provides an interface for managing dialog boxes, the Visual C++ dialog editor makes it easy to design dialog boxes and create their dialog-template resources, and Code wizards simplify the process of initializing and validating the controls in a dialog box and of gathering the values entered by the user.

Dialog boxes contain controls, including:

- Windows common controls such as edit boxes, pushbuttons, list boxes, combo boxes, tree controls, list controls, and progress indicators.
- ActiveX controls.
- Owner-drawn controls: controls that you are responsible for drawing in the dialog box.

Most dialog boxes are modal, which require the user to close the dialog box before using any other part of the program. But it is possible to create modeless dialog boxes, which let users work with other windows while the dialog box is open. MFC supports both kinds of dialog box with class `CDialog`. The controls are arranged and managed using a dialog-template resource, created with the dialog editor.

Property sheets, also known as tab dialog boxes, are dialog boxes that contain "pages" of distinct dialog-box controls. Each page has a file folder "tab" at the top. Clicking a tab brings that page to the front of the dialog box.

What do you want to know more about

- Example: Displaying a Dialog Box via a Menu Command
- Dialog-box components in the framework
- Modal and modeless dialog boxes
- Property sheets and property pages in a dialog box
- Creating the dialog resource
- Creating a dialog class with Code Wizards
- Life cycle of a dialog box
- Dialog data exchange (DDX) and validation (DDV)
- Type-safe access to controls in a dialog box
- Mapping Windows messages to your class
- Commonly Overridden Member Functions
- Commonly Added Member Functions
- Common dialog classes
- Dialog boxes in OLE
- Create an application whose user interface is a dialog box: see the CMNCTRL1 or CMNCTRL2 sample programs. The Application Wizard provides this option as well.
- Samples

See Also

User Interface Elements

Property Sheets and Property Pages (MFC)

The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.

The latest version of this topic can be found at Property Sheets and Property Pages (MFC).

An MFC dialog box can take on a "tab dialog" look by incorporating property sheets and property pages. Called a "property sheet" in MFC, this kind of dialog box, similar to many dialog boxes in Microsoft Word, Excel, and Visual C++, appears to contain a stack of tabbed sheets, much like a stack of file folders seen from front to back, or a group of cascaded windows. Controls on the front tab are visible; only the labeled tab is visible on the rear tabs. Property sheets are particularly useful for managing large numbers of properties or settings that fall fairly neatly into several groups. Typically, one property sheet can simplify a user interface by replacing several separate dialog boxes.

As of MFC version 4.0, property sheets and property pages are implemented using the common controls that come with Windows 95 and Windows NT version 3.51 and later.

Property sheets are implemented with classes CPropertySheet and CPropertyPage(described in the *MFC Reference*). `CPropertySheet` defines the overall dialog box, which can contain multiple "pages" based on `CPropertyPage`.

For information on creating and working with property sheets, see the topic Property Sheets.

See Also

Dialog Boxes
Life Cycle of a Dialog Box
Property Sheets and Property Pages in MFC
Exchanging Data
Creating a Modeless Property Sheet
Handling the Apply Button

Handling the Apply Button

The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.

The latest version of this topic can be found at Handling the Apply Button.

Property sheets have a capability that standard dialog boxes do not: They allow the user to apply changes they have made before closing the property sheet. This is done using the Apply button. This article discusses methods you can use to implement this feature properly.

Modal dialog boxes usually apply the settings to an external object when the user clicks OK to close the dialog box. The same is true for a property sheet: When the user clicks OK, the new settings in the property sheet take effect.

However, you may want to allow the user to save settings without having to close the property sheet dialog box. This is the function of the Apply button. The Apply button applies the current settings in all of the property pages to the external object, as opposed to applying only the current settings of the currently active page.

By default, the Apply button is always disabled. You must write code to enable the Apply button at the appropriate times, and you must write code to implement the effect of Apply, as explained below.

If you do not wish to offer the Apply functionality to the user, it is not necessary to remove the Apply button. You can leave it disabled, as will be common among applications that use standard property sheet support available in future versions of Windows.

To report a page as being modified and enable the Apply button, call **CPropertyPage::SetModified(TRUE)**. If any of the pages report being modified, the Apply button will remain enabled, regardless of whether the currently active page has been modified.

You should call CPropertyPage::SetModified whenever the user changes any settings in the page. One way to detect when a user changes a setting in the page is to implement change notification handlers for each of the controls in the property page, such as **EN_CHANGE** or **BN_CLICKED**.

To implement the effect of the Apply button, the property sheet must tell its owner, or some other external object in the application, to apply the current settings in the property pages. At the same time, the property sheet should disable the Apply button by calling **CPropertyPage::SetModified(FALSE)** for all pages that applied their modifications to the external object.

For an example of this process, see the MFC General sample PROPDLG.

See Also

Property Sheets

Visual C++ Samples

The Visual C++ samples listed below demonstrate different functionalities across multiple technologies.

Visual Studio samples

Windows Store app samples

The All-In-One code framework

Windows Desktop code samples

MFC samples

CodePlex samples

MFC samples on CodePlex

ADO code samples

Windows Hardware development samples

STL Samples

Important

This sample code is intended to illustrate a concept, and it shows only the code that is relevant to that concept. It may not meet the security requirements for a specific environment, and it should

not be used exactly as shown. We recommend that you add security and error-handling code to make your projects more secure and robust. Microsoft provides this sample code "AS IS" with no warranties.

See Also

Visual C++ Reference

Exchanging Data

As with most dialog boxes, the exchange of data between the property sheet and the application is one of the most important functions of the property sheet. This article describes how to accomplish this task.

Exchanging data with a property sheet is actually a matter of exchanging data with the individual property pages of the property sheet. The procedure for exchanging data with a property page is the same as for exchanging data with a dialog box, since a CPropertyPage object is just a specialized CDialog object. The procedure takes advantage of the framework's dialog data exchange (DDX) facility, which exchanges data between controls in a dialog box and member variables of the dialog box object.

The important difference between exchanging data with a property sheet and with a normal dialog box is that the property sheet has multiple pages, so you must exchange data with all the pages in the property sheet. For more information on DDX, see Dialog Data Exchange and Validation.

The following example illustrates exchanging data between a view and two pages of a property sheet:

```
void CMyView::DoModalPropertySheet()

{

   CPropertySheet propsheet;

   CMyFirstPage pageFirst; // derived from CPropertyPage

   CMySecondPage pageSecond; // derived from CPropertyPage



   // Move member data from the view (or from the currently

   // selected object in the view, for example).

   pageFirst.m_nMember1 = m_nMember1;

   pageFirst.m_nMember2 = m_nMember2;



   pageSecond.m_strMember3 = m_strMember3;
```

```
  pageSecond.m_strMember4 = m_strMember4;


  propsheet.AddPage(&pageFirst);
  propsheet.AddPage(&pageSecond);


  if (propsheet.DoModal() == IDOK)
  {
    m_nMember1 = pageFirst.m_nMember1;
    m_nMember2 = pageFirst.m_nMember2;
    m_strMember3 = pageSecond.m_strMember3;
    m_strMember4 = pageSecond.m_strMember4;
    GetDocument()->SetModifiedFlag();
    GetDocument()->UpdateAllViews(NULL);
  }
}
```

MFC Desktop Applications

The new home for Visual Studio documentation is [Visual Studio 2017 Documentation] on docs.microsoft.com.

The latest version of this topic can be found at [MFC Desktop Applications].

The Microsoft Foundation Class (MFC) Library provides an object-oriented wrapper over much of the Win32 and COM APIs. Although it can be used to create very simple desktop applications, it is most useful when you need to develop more complex user interfaces with multiple controls. You can use MFC to create applications with Office-style user interfaces.

The MFC Reference covers the classes, global functions, global variables, and macros that make up the Microsoft Foundation Class Library.

The individual hierarchy charts included with each class are useful for locating base classes. The MFC Reference usually does not describe inherited member functions or inherited operators. For information on these functions, refer to the base classes depicted in the hierarchy diagrams.

The documentation for each class includes a class overview, a member summary by category, and topics for the member functions, overloaded operators, and data members.

Public and protected class members are documented only when they are normally used in application programs or derived classes. See the class header files for a complete listing of class members.

Important

The MFC classes and their members cannot be used in applications that execute in the Windows Runtime.

MFC libraries (DLLs) for multibyte character encoding (MBCS) are no longer included in Visual Studio, but are available as a Visual Studio add-on. For more information, see **MFC MBCS DLL Add-on**.

In This Section

Concepts
Conceptual articles on MFC topics.

Hierarchy Chart
Visually details the class relationships in the class library.

Class Overview
Lists the classes in the MFC Library according to category.

Walkthroughs
Contains articles that walk you through various tasks associated with MFC library features.

Technical Notes
Provides links to specialized topics, written by the MFC development team, on the class library.

Customization for MFC
Provides some tips for customizing your MFC application.

Classes
Provides links to and header file information for the MFC classes.

Internal Classes
Used internally in MFC. For completeness, this section describes these internal classes, but they are not intended to be used directly in your code.

Macros and Globals
Provides links to the macros and global functions in the MFC Library.

Structures, Styles, Callbacks, and Message Maps
Provides links to the structures, styles, callbacks, and message maps used by the MFC Library.

MFC Wizards and Dialog Boxes
A guide to the features in Visual Studio for creating MFC applications.

Working with Resource Files
How to use resource files to manage static user interface data such as UI strings and dialog box layout.

Related Sections

Hierarchy Chart Categories
Describes the MFC hierarchy chart by category.

ATL/MFC Shared Classes
Provides links to classes that are shared between MFC and ATL.

MFC Samples
Provides links to samples that demonstrate how to use MFC.

Visual C++ Libraries Reference
Provides links to the various libraries provided with Visual C++, including ATL, MFC, OLE DB Templates, the C run-time library, and the Standard C++ Library.

Debugging in Visual Studio
Provides links to using the Visual Studio debugger to correct logic errors in your application or stored procedures.

See Also

MFC and ATL

Class Library Overview

The new home for Visual Studio documentation is Visual Studio 2017 Documentation on docs.microsoft.com.

The latest version of this topic can be found at Class Library Overview.

This overview categorizes and describes the classes in the Microsoft Foundation Class Library (MFC) version 9.0. The classes in MFC, taken together, constitute an application framework — the framework of an application written for the Windows API. Your programming task is to fill in the code that is specific to your application.

The library's classes are presented here in the following categories:

- Root Class: CObject
- MFC Application Architecture Classes
  - Application and Thread Support Classes
  - Command Routing Classes
  - Document Classes
  - View Classes (Architecture)
  - Frame Window Classes (Architecture)
  - Document-Template Classes
- Window, Dialog, and Control Classes
  - Frame Window Classes (Windows)

- View Classes (Windows)
- Dialog Box Classes
- Control Classes
- Control Bar Classes
- Drawing and Printing Classes
  - Output (Device Context) Classes
  - Drawing Tool Classes
- Simple Data Type Classes
- Array, List, and Map Classes
  - Template Classes for Arrays, Lists, and Maps
  - Ready-to-Use Array Classes
  - Ready-to-Use List Classes
  - Ready-to-Use Map Classes
- File and Database Classes
  - File I/O Classes
  - DAO Classes
  - ODBC Classes
  - OLE DB Classes
- Internet and Networking Classes
  - Windows Sockets Classes
  - Win32 Internet Classes
- OLE Classes
  - OLE Container Classes
  - OLE Server Classes
  - OLE Drag-and-Drop and Data Transfer Classes
  - OLE Common Dialog Classes
  - OLE Automation Classes
  - OLE Control Classes
  - Active Document Classes
  - OLE-Related Classes
- Debugging and Exception Classes
  - Debugging Support Classes
  - Exception Classes

The section General Class Design Philosophy explains how the MFC Library was designed.

For an overview of the framework, see Using the Classes to Write Applications for Windows. Some of the classes listed above are general-purpose classes that can be used outside of the framework and provide useful abstractions such as collections, exceptions, files, and strings.

To see the inheritance of a class, use the Class Hierarchy Chart.

In addition to the classes listed in this overview, the MFC Library contains a number of global functions, global variables, and macros. There is an overview and detailed listing of these in the topic MFC Macros and Globals, which follows the alphabetical reference to the MFC classes.

See Also

MFC Desktop Applications