27

MULTIPLE DOCUMENT INTERFACE (MDI)

Many programs deal with only one file or document at a time. These applications are classified as Single Document Interface, or SDI, applications. An example of an SDI application is the Windows Notepad applet. However, users often need to see multiple documents at a time, or be able to present multiple views into the same document. Applications that let the user do this are called Multiple Document Interface applications.

An MDI application consists of three types of windows. The main window of the application is known as the MDI frame window. It consists of the title bar, sizing border, system menu, minimize button, and other system-defined features. An MDI application registers a window class and provides a window procedure for the MDI frame window, just as it would for a normal application window. An MDI application does not display output in its MDI frame window's client area, however. Instead, it creates an MDI client window that occupies the client area of the MDI frame window. The MDI client window belongs to the preregistered window class MDICLIENT. The MDI client window supports the creation and management of individual MDI child windows within which the document information is displayed. An MDI application may wish to display different types of document information, and therefore may contain different types of MDI child windows. Figure 27-1 shows a simple MDI application, and identifies the MDI frame window, client window, and child window.

Creating an MDI Application

The first step in creating an MDI application is to register your window classes. You will need a window class for the MDI frame window, and a window class for each different type of MDI child window that your application supports. The class structure for an MDI frame window is filled in like the class structure for an SDI application's main window. The class structure for an MDI child window is filled in like the class structure for child windows in SDI applications, with two exceptions. First, the class structure for an MDI child window should specify an ICON, since the user can minimize MDI child windows within the MDI frame window. Second, the menu name should be NULL, since an MDI child window does not have its own menu. The following example illustrates the registration of two window classes. MyMDIApp is the class name of an MDI frame window class, and MDIChild is the class name of an MDI child window class.

```
LPCTSTR lpszAppName  = "MyMDIApp";
```

```
LPCTSTR lpszChild    = "MDIChild";
LPCTSTR lpszTitle    = "MDI Test Application";

int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow )
{
   MSG      msg;
   HWND     hWnd;
   WNDCLASS wc;

   // Register the main application window class.
   //.........................................
   wc.style         = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc   = (WNDPROC)WndProc;
   wc.cbClsExtra    = 0;
   wc.cbWndExtra    = 0;
   wc.hInstance     = hInstance;
   wc.hIcon         = LoadIcon( hInstance, lpszAppName );
   wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = (HBRUSH)(COLOR_APPWORKSPACE+1);
   wc.lpszMenuName  = lpszAppName;
   wc.lpszClassName = lpszAppName;

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );

   // Register the window class for the MDI child windows.
   //.....................................................
   wc.lpfnWndProc   = (WNDPROC)ChildWndProc;
   wc.hIcon         = LoadIcon( hInstance, lpszChild );
   wc.hCursor       = LoadCursor( NULL, IDC_ARROW );
   wc.hbrBackground = (HBRUSH)(COLOR_WINDOW+1);
   wc.lpszMenuName  = NULL;
   wc.lpszClassName = lpszChild;

   if ( IS_WIN95 )
   {
      if ( !RegisterWin95( &wc ) )
         return( FALSE );
   }
   else if ( !RegisterClass( &wc ) )
      return( FALSE );
         .
         .
         .

BOOL RegisterWin95( CONST WNDCLASS* lpwc )
{
   WNDCLASSEX wcex;

   wcex.style         = lpwc->style;
   wcex.lpfnWndProc   = lpwc->lpfnWndProc;
   wcex.cbClsExtra    = lpwc->cbClsExtra;
   wcex.cbWndExtra    = lpwc->cbWndExtra;
   wcex.hInstance     = lpwc->hInstance;
   wcex.hIcon         = lpwc->hIcon;
   wcex.hCursor       = lpwc->hCursor;
   wcex.hbrBackground = lpwc->hbrBackground;
   wcex.lpszMenuName  = lpwc->lpszMenuName;
   wcex.lpszClassName = lpwc->lpszClassName;

   // Added elements for Windows 95.
   //...............................
   wcex.cbSize = sizeof(WNDCLASSEX);
   wcex.hIconSm = LoadImage(wcex.hInstance, lpwc->lpszClassName,
                            IMAGE_ICON, 16, 16,
                            LR_LOADREALSIZE );
```

```
   return RegisterClassEx( &wcex );
}
```

After you register the window classes, create the MDI frame window just as you would in an SDI application.

```
         .
         .
         .
   // Create the main application window.
   //.................................
   hWnd = CreateWindow( lpszAppName,
                        lpszTitle,
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        NULL,
                        NULL,
                        hInstance,
                        NULL
                      );

   if ( !hWnd )
      return( FALSE );

   ShowWindow( hWnd, nCmdShow );
   UpdateWindow( hWnd );
         .
         .
         .
```

MDI Applications must call the TranslateMDISysAccel() function in the main message loop to process the pre-defined MDI specific accelerator keys. For MDI applications that have an accelerator table, you must call the TranslateMDISysAccel() function before the call to TranslateAccelerator() in order to allow Windows to handle any pre-defined MDI accelerators before the application specific accelerators. Since this test application does not have an accelerator table, only the TranslateMDISysAccel() function is called as shown in the following code segment:

```
         .
         .
         .
   while( GetMessage( &msg, NULL, 0, 0) )
   {
      if ( hWndClient && TranslateMDISysAccel( hWndClient, &msg ) )
         continue;

      TranslateMessage( &msg );
      DispatchMessage( &msg );
   }
         .
         .
         .
```

Once the main message loop for your application is running, it becomes the responsibility of the MDI frame window's window procedure to coordinate the creation and management of the individual MDI child windows.

The MDI Frame Window

The MDI frame window is responsible for creating the MDI client window, usually during processing of the WM_CREATE message. You create the MDI client window using the preregistered window class MDICLIENT.

The MDI client window will dynamically alter one of the pop-up menu items in the frame window's menu bar, so you must pass the menu handle for the pop-up menu that the MDI client window is to alter as the final parameter to the CreateWindowEx() function. For more detailed information on how the MDI client window alters this pop-up menu, refer to Menus in MDI applications later in this chapter.

```
HWND  hWndClient = NULL;

// An ID that is different than all menu ids.
```

```
//.....................................
#define ID_CHILDWINDOW 1000

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_CREATE :
              {
                 CLIENTCREATESTRUCT ccs;

                 // Assign the 'Window' menu.
                 //.........................
                 ccs.hWindowMenu  = GetSubMenu( GetMenu( hWnd ), 1 );
                 ccs.idFirstChild = ID_CHILDWINDOW;

                 // Create the client window.
                 //.........................
                 hWndClient = CreateWindowEx( WS_EX_CLIENTEDGE,
                                              "MDICLIENT", NULL,
                                              WS_CHILD | WS_CLIPCHILDREN,
                                              0, 0, 0, 0,
                                              hWnd, (HMENU)0xCA0, hInst, &ccs);

                 ShowWindow( hWndClient, SW_SHOW );
              }
              break;
        .
        .
        .
```

One additional style bit is available when creating the MDI client window. If you create the MDI client with the MDIS_ALLCHILDSTYLES style bit, Windows will not limit the valid style bits for MDI child windows. If the MDI client window is created without the MDIS_ALLCHILDSTYLES style bit set, then only those values specified in Table 27-1 may be specified. Use of the MDIS_ALLCHILDSTYLES style bit allows an application to create MDI child windows with nonstandard MDI child behaviors.

Table 27-1. dwStyle Parameter Values in CreateMDIWindow()

Value	Description
WS_MINIMIZE	Window is created minimized.
WS_MAXIMIZE	Window is created maximized.
WS_HSCROLL	Window has a horizontal scrollbar.
WS_VSCROLL	Window has a vertical scrollbar.

The MDI frame window also is responsible for creating and destroying the individual MDI child windows. You create MDI child windows using either the CreateMDIWindow() function or the WM_MDICREATE window message. This is typically done in response to menu items such as File Open, File New, File Save, and File Close. The following example, taken from the window procedure of the MDI frame window, illustrates the creation of an MDI child window in response to the menu item IDM_FILE_NEW.

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_NEW :
                     {
                        HWND hWndChild;

                        // Create a new child window.
```

```
                    //.........................
                    hWndChild = CreateMDIWindow( (LPTSTR)lpszChild,
                                    "Document", 0L,
                                    CW_USEDEFAULT, CW_USEDEFAULT,
                                    CW_USEDEFAULT, CW_USEDEFAULT,
                                    hWndClient, hInst, 0L);

                    ShowWindow( hWndChild, SW_SHOW );
                }
                break;
        .
        .
        .
```

In all cases where the MDI frame window procedure does not explicitly handle a particular message, that message must be passed to the DefFrameProc() function, as opposed to the DefWindowProc() function in a normal window procedure. This allows Windows to provide default MDI behavior.

The MDI Child Window

The work that would normally be done in an SDI application's main window is carried out in MDI applications by the MDI child window. The window procedure for an MDI child window is identical to its SDI counterpart, with the exception of default message processing. An MDI child window uses the function DefMDIChildProc() to handle unprocessed window messages.

The major philosophical difference between an SDI application window and an MDI child window is that the SDI application contains only one instance of the application window, which is used to manipulate one set of data. An MDI application will usually have several instances of the MDI child window, all acting on different sets of data. Note, however, that there is only one window procedure specified for all instances of a given MDI child window. This means that the code in the window procedure for an MDI child window must be able to determine which instance of the data the current message is for.

The window procedure is passed the window handle of the MDI child window as its first parameter. You can use one of several techniques to associate a particular set of data with this window handle.

The CreateMDIWindow() function and the WM_MDICREATE message allow the creating procedure, typically an MDI frame window, to pass a 32-bit value to the MDI child window during the processing of its WM_CREATE message. This 32-bit value could contain a pointer to the data structure that this MDI child window is to access. The MDI child window could then store this pointer in one of the window's extra bytes. Allocate space for the data by specifying a value that is the number of bytes required to contain a pointer to your data structure for the cbWndExtra member of the WNDCLASS structure when registering the window class. The MDI child window then can use GetWindowLong(), casting as appropriate, to retrieve this pointer. The following example illustrates how an MDI child window would retrieve this 32-bit value during processing of its WM_CREATE message.

```
LRESULT ChildWndProc( HWND hWndChild, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
static LPARAM FrameParam;

    switch(uMsg)
    {
        case WM_CREATE :
              {
                 LPCREATESTRUCT    lpCreateStruct    = (LPCREATESTRUCT)lParam;
                 LPMDICREATESTRUCT lpMDICreateStruct = (LPMDICREATESTRUCT)lpCreateStruct->lpCreateParams;

                 FrameParam  = lpMDICreateStruct->lParam;
              }
              break;
        .
        .
        .
```

An application can use window properties instead of using the window's extra bytes. The advantage to window properties is that you do not need to allocate extra space for the data when registering the window class. In addition, window properties are accessed using strings, which can be self-documenting. In this case, the value of the window property would be a pointer to the data structure associated with the given window.

Finally, some applications may find it more appropriate to keep individual data structures in a linked list, and to include the window handle of the window associated with a given data structure as part of the structure itself. It is then necessary only to traverse the linked list, matching window handles in the data structure against the window handle given to the window procedure to determine for which data structure a given message is intended.

Menus in MDI Applications

The MDI frame window menu should include a window pop-up menu item. This pop-up menu item is typically defined just before the Help pop-up menu item, and contains submenu items used to Tile, Cascade, Arrange Icons, and Close All Child Windows. These are implemented using the window messages defined in the Message Summary section at the end of this chapter. In addition, the MDI client window will add the names of newly created MDI child windows to the bottom of the pop-up menu item you specify when creating the MDI client window.

Figure 27-2 shows a simple MDI application. There are currently four MDI child windows open, one of which has been minimized to the bottom of the MDI client window. Notice the Window menu. The MDI client window has dynamically placed the names of the four MDI child windows at the bottom of the pop-up menu, and has placed a checkmark next to the currently active MDI child window.

Windows provides several accelerator keys for MDI applications. It requires no extra code to implement these accelerators. These functions are provided simply by using the MDI versions of the default message-processing functions in the MDI frame and MDI child windows, using the MDI accelerator translation function in your main message loop, and having an MDI client window of the MDICLIENT class. Table 27-2 defines these new accelerator keys.

Table 27-2. Accelerator Keys for MDI Applications

Accelerator Key	Purpose
A-a-0	Opens the MDI child windows system menu.
A-4	Closes the active MDI child window.
A-6	Activates the next MDI child window.
A-F-6	Activates the previous MDI child window.

Table 27-3 specifies the Windows functions used to implement the MDI application interface. Detailed descriptions of each function follow the table.

Table 27-3. MDI Function Summary.

Function	Purpose
ArrangeIconicWindows	Arranges minimized MDI child windows at the bottom of the MDI client window.
CascadeWindows	Arranges nonminimized MDI child windows in a cascade (overlapped) arrangement.
CreateMDIWindow	Creates a new MDI child window and returns its window handle.
DefFrameProc	Used in the window procedure of an MDI frame window to process any unhandled window messages.
DefMDIChildProc	Used in the window procedure of an MDI child window to process any unhandled window messages.
TileWindows	Arranges nonminimized MDI child windows in a tiled (nonoverlapped) arrangement.
TranslateMDISysAccel	Used in the main message loop of an MDI application to handle any pre-defined window accelerators specific to MDI windows.
Messages	
WM_MDIACTIVATE	When sent to an MDI client window, causes a new MDI child window to be activated. When received by an MDI child window, indicates that the window is either being activated or deactivated.
WM_MDICASCADE	When sent to an MDI client window, causes all of the MDI child windows to be cascaded.

WM_MDICREATE	When sent to an MDI client window, creates a new MDI child window.
WM_MDIDESTROY	Indicates to an MDI child window that it is being destroyed.
WM_MDIGETACTIVE	When sent to an MDI client window, returns the window handle of the currently active MDI child window.
WM_MDIICONARRANGE	When sent to an MDI client window, causes all of the minimized MDI child windows icons to be arranged along the bottom of the MDI client window.
WM_MDIMAXIMIZE	When sent to an MDI client window, causes the specified MDI child window to be maximized.
WM_MDINEXT	When sent to an MDI client window, causes the MDI child window after (or before) the specified MDI child window to become active.
WM_MDIREFRESHMENU	When sent to an MDI client window, causes the MDI client to update the state of the menu after changes have been made.
WM_MDIRESTORE	When sent to an MDI client window, causes the indicated MDI child window to be restored from either a maximized or minimized state.
WM_MDISETMENU	When sent to an MDI client window, sets the MDI frame menu to the indicated menu. This allows MDI child windows to implement individual menus.
WM_MDITILE	When sent to an MDI client window, causes all MDI child windows to be tiled.

ARRANGEICONICWINDOWS

WIN32S WINDOWS 95 WINDOWS NT

Description	An application can use ArrangeIconicWindows() to arrange all minimized MDI child windows. An application also can use the WM_MDIICONARRANGE message as an alternative. Windows will arrange the minimized MDI child windows along the bottom of the MDI client window.
Syntax	UINT ArrangeIconicWindows(HWND hWndClient)
Parameters	
hWndClient	HWND: Specifies the window handle of the MDI client window that is to manage this MDI child window.
Returns	If successful, this function returns the height of one row of icons; otherwise, it returns zero.
Include File	winuser.h
See Also	WM_MDIICONARRANGE
Example	The following code segment would appear as part of the window procedure for the MDI frame window, responding to a menu item with the identifier IDM_ARRANGE.

```
HWND  hWndClient;

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_ARRANGE :
                        ArrangeIconicWindows( hWndClient );
                        break;
        .
        .
        .
```

CASCADEWINDOWS

WINDOWS 95

Description	An application uses CascadeWindows() to arrange all nonminimized MDI child windows in a cascaded (overlapped) manner. CascadeWindows() is similar to the WM_MDICASCADE message and identical to the CascadeChildWindows() function, except that the user can specify the window handles of the windows to be affected, and can limit the area of the MDI client window that is used.

Syntax	WORD API CascadeWindows(HWND hWndParent, WORD wFlags, LPCRECT lpWindowRect, WORD nWindowCount, CONST HWND *lpWndArray)
Parameters	
hWndParent	HWND: Specifies the window handle of the parent window. Child windows of the parent window will be arranged. This parameter typically specifies the MDI client window handle. If this parameter is NULL, the desktop window is assumed.
wFlags	WORD: Specifies various arrangement flags. Table 27-4 gives a list of valid values.
lpWindowRect	LPCRECT: Points to a RECT structure that defines the boundaries within which the child windows are arranged. If this parameter is NULL, the entire client area of the parent window is assumed.
nWindowCount	WORD: Specifies the number of window handles included in the lpWndArray array. This value is ignored if lpWndArray is NULL.
lpWndArray	CONST HWND *: Specifies an array of child window handles. If this parameter is NULL, then all child windows of the parent window are subject to being arranged.
Returns	The return value is the number of windows arranged. If the function fails, the return value is zero.
Include File	winuser.h
See Also	CascadeChildWindows(), TileChildWindows(), TileWindow()
Related Messages	WM_MDICASCADE
Example	The following code segment responds the the Cascade menu item. If the application is running on Windows 95, the CascadeWindows() function is called. If the application is running on Windows NT, the WM_MDICASCADE message is sent to the MDI client window.

```
HWND  hWndClient;

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
               switch( LOWORD( wParam ) )
               {
                  case IDM_CASCADE :
                         if ( IS_WIN95 )
                            CascadeWindows( hWndClient, MDITILE_SKIPDISABLED,
                                            NULL, 0, NULL );
                         else
                            SendMessage( hWndClient, WM_MDICASCADE,
                                         MDITILE_SKIPDISABLED, 0 );
                         break;
         .
         .
         .
```

CREATEMDIWINDOW

WIN32S WINDOWS 95 WINDOWS NT

Description	An application must use CreateMDIWindow() to create MDI child windows, rather than using the CreateWindow() or CreateWindowEx() functions. An application also can use the WM_MDICREATE message as an alternative.
Syntax	HWND CreateMDIWindow(LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, int x, int y, int nWidth, int nHeight, HWND hWndClient, HINSTANCE hInstance, LPARAM lParam)
Parameters	
lpszClassName	LPCTSTR: Points to a null-terminated string specifying the class name of the MDI child window. The class name must have been registered by a call to RegisterClass().

lpszWindowName	LPCTSTR: Points to a null-terminated string that specifies the window name. The window name is displayed in the title bar of the MDI child window.
dwStyle	DWORD: Specifies the window styles of the MDI child window. If the MDI client was created specifying the style MDIS_ALLCHILDSTYLES, then any window styles listed in the discussion of CreateWindow() can be specified. Otherwise, only the styles shown in Table 27-1 may be specified.
x	int: Specifies the initial horizontal position, in client coordinates, of the MDI child window. Use the value CW_USEDEFAULT to allow Windows to assign a default value to this parameter.
y	int: Specifies the initial vertical position, in client coordinates, of the MDI child window. Use the value CW_USEDEFAULT to allow Windows to assign a default value to this parameter.
nWidth	int: Specifies the initial width, in client coordinates, of the MDI child window. Use the value CW_USEDEFAULT to allow Windows to assign a default value to this parameter.
nHeight	int: Specifies the initial height, in client coordinates, of the MDI child window. Use the value CW_USEDEFAULT to allow Windows to assign a default value to this parameter.
hWndClient	HWND: Specifies the window handle of the MDI client window that is to manage this MDI child window.
hInstance	HINSTANCE: Identifies the instance of the application that is creating the MDI child window.
lParam	LPARAM: An application-defined value. This value is passed to the MDI child window during the processing of the WM_CREATE message. When the MDI child window receives a WM_CREATE message, the lParam of this message contains a pointer to a CREATESTRUCT structure. The first member of this structure, lpCreateParams, contains a pointer to an MDICREATESTRUCT structure. The lParam member of this structure contains the 32-bit value of lParam from the CreateMDIWindow() function. See the definition of the CREATESTRUCT structure below. See the definition of the MDICREATESTRUCT structure under the WM_MDICREATE message.
Returns	If successful, this function returns the window handle of the new MDI child window; otherwise, it returns NULL.
Include File	winuser.h
See Also	WM_MDICREATE

CREATESTRUCT Definition

```
typedef struct tagCREATESTRUCT
{
   LPVOID    lpCreateParams;
   HINSTANCE hInstance;
   HMENU     hMenu;
   HWND      hwndParent;
   int       cy;
   int       cx;
   int       y;
   int       x;
   LONG      style;
   LPCTSTR   lpszName;
   LPCTSTR   lpszClass;
   DWORD     dwExStyle;
} CREATESTRUCT;
```

lpCreateParams	LPVOID: A pointer to data to be used for creating the window. In Windows NT, this member is the address of a SHORT(16-bit) value that specifies the size, in bytes, of the window creation data. This value is followed by the creation data. When referring to the data pointed to by this parameter, because the pointer may not be DWORD aligned, the application should use a pointer declared as UNALIGNED.
hInstance	HINSTANCE: The instance handle of the module that owns the new window.
hMenu	HMENU: The menu to be used by the new window.
hwndParent	HWND: The parent window handle, if the window is a child window. If the window is owned, this member identifies the owner window. If the window is not a child or owned window, this member is NULL.

cy	int: The height, in pixels, of the new window.
cx	int: The width, in pixels, of the new window.
y	int: The y-coordinate of the upper left corner of the new window. If the new window is a child window, coordinates are relative to the parent window. Otherwise, the coordinates are relative to the screen origin.
x	int: The x-coordinate of the upper left corner of the new window. If the new window is a child window, coordinates are relative to the parent window. Otherwise, the coordinates are relative to the screen origin.
style	LONG: The style for the new window. This is a combination of the window styles valid for use with the CreateWindow() function.
lpszName	LPCTSTR: A pointer to a null-terminated string that specifies the name of the new window.
lpszClass	LPCTSTR: A pointer to a null-terminated string that specifies the class name of the new window.
dwExStyle	DWORD: The extended style for the new window. This ia combination of the extended window styles valid for use with the CreateWindowEx() function.
Example	The code segment creates a new MDI child window when the user selects the New menu item. The currently active MDI child window is closed when the user selects the Close menu item.

```
HWND  hWndClient;

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
              switch( LOWORD( wParam ) )
              {
                 case IDM_NEW :
                     {
                        HWND hWndChild;

                        // Create a new child window.
                        //..........................
                        hWndChild = CreateMDIWindow( (LPTSTR)lpszChild,
                                        "Document", 0L,
                                        CW_USEDEFAULT, CW_USEDEFAULT,
                                        CW_USEDEFAULT, CW_USEDEFAULT,
                                        hWndClient, hInst, 0L);

                        ShowWindow( hWndChild, SW_SHOW );
                     }
                     break;

                 case IDM_CLOSE :
                     {
                        HWND hActiveWnd;

                        // Close the active child window.
                        //...............................
                        hActiveWnd = (HWND)SendMessage( hWndClient,
                                                WM_MDIGETACTIVE, 0, 0 );
                        if ( hActiveWnd )
                           SendMessage( hWndClient, WM_MDIDESTROY, (WPARAM)hActiveWnd, 0 ); }
                     break;

                 default :
                        return( DefFrameProc( hWnd, hWndClient, uMsg, wParam, lParam ) );
              }
              break;

      case WM_DESTROY :
              PostQuitMessage(0);
              break;
```

```
         .
         .
         .

      default :
            return( DefFrameProc( hWnd, hWndClient, uMsg, wParam, lParam ) );
   }

   return( 0L );
}
```

DEFFRAMEPROC

WIN32S WINDOWS 95 WINDOWS NT

Description	The window procedure of an MDI frame window passes all unprocessed messages to DefFrameProc(). DefFrameProc() allows Windows to provide default MDI application behavior.
Syntax	LRESULT DefFrameProc(HWND hWndFrame, HWND hWndClient, UINT uMsg, WPARAM wParam, LPARAM lParam)
Parameters	
hWndFrame	HWND: Specifies the window handle of the MDI frame window.
hWndClient	HWND: Specifies the window handle of the MDI client window.
uMsg	UINT: Specifies the message to be processed.
wParam	WPARAM: Specifies additional data, specific to this message.
lParam	LPARAM: Specifies additional data, specific to this message.
Returns	The return value is specific to the message being processed. The MDI frame windows window procedure should simply return Delframe Prac()'s the return value.
Include File	winuser.h
See Also	DefMDIChildProc(), DefWindowProc()
Example	See the example for the CreateMDIWindow() function.

DEFMDICHILDPROC

WIN32S WINDOWS 95 WINDOWS NT

Description	The window procedure of an MDI child window passes all unprocessed messages to this function. DefMDIChildProc() allows Windows to provide default MDI application behavior.
Syntax	LRESULT DefMDIChildProc(HWND hWndChild, UINT uMsg, WPARAM wParam, LPARAM lParam)
Parameters	
hWndChild	HWND: Specifies the window handle of the MDI child window.
uMsg	UINT: Specifies the message to be processed.
wParam	WPARAM: Specifies additional data, specific to this message.
lParam	LPARAM: Specifies additional data, specific to this message.
Returns	The return value is specific to the message being processed. The MDI child window's window procedure should simply return DefMDIChildProc()'s the return.
Include File	winuser.h
See Also	DefFrameProc(), DefWindowProc()
Example	The following code segment shows the implementation of a minimal window procedure for an MDI child window.

```
LRESULT CALLBACK ChildWndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      default :
```

```
            return( DefMDIChildProc( hWnd, uMsg, wParam, lParam ) );
   }

   return( 0L );
}
```

TILEWINDOWS — WINDOWS 95

Description	An application uses TileWindows() to arrange all nonminimized MDI child windows in a tiled (nonoverlapped) manner. This function is similar to the WM_MDITILE message and identical to the TileChildWindows() function, except that the user can specify the window handles of the windows to be affected, and can limit the area of the MDI client window that is used.
Syntax	WORD API TileWindows(HWND hWndParent, WORD wFlags, LPCRECT lpWindowRect, WORD nWindowCount, CONST HWND *lpWndArray)
Parameters	
hWndParent	HWND: Specifies the window handle of the parent window. Child windows of the parent window will be arranged. This parameter typically specifies the MDI client window handle. If this parameter is NULL, the desktop window is assumed.
wFlags	WORD: Specifies various arrangement flags. Table 27-4 gives a list of valid values. In addition, the MDITILE_VERTICAL and MDITILE_HORIZONTAL flags from Table 27-6 may be specified.
lpWindowRect	LPCRECT: Points to a RECT structure that defines the boundaries within which the child windows are arranged. If this parameter is NULL, the entire client area of the parent window is assumed.
nWindowCount	WORD: Specifies the number of window handles included in the lpWndArray array. This value is ignored if lpWndArray is NULL.
lpWndArray	CONST HWND *: Specifies an array of child window handles. If this parameter is NULL, then all child windows of the parent window are subject to being arranged.
Returns	The return value is the number of windows arranged. If the function fails, the return value is zero.
Include File	winuser.h
See Also	CascadeChildWindows(),CascadeWindows(), TileChildWindows()
Related Messages	WM_MDITILE
Example	This code segment responds to the user selecting the Tile Horizontally and Tile Vertically menu items. The TileWindows() function is used if the application is running on Windows 95, otherwise, it sends the WM_MDITILE message to the client window.

```
HWND  hWndClient;

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
   switch( uMsg )
   {
      case WM_COMMAND :
               switch( LOWORD( wParam ) )
               {
                  case IDM_TILEHORZ :
                         if ( IS_WIN95 )
                            TileWindows( hWndClient, MDITILE_HORIZONTAL,
                                         NULL, 0, NULL );
                         else
                            SendMessage( hWndClient, WM_MDITILE,
                                         MDITILE_HORIZONTAL, 0 );
                         break;

                  case IDM_TILEVERT :
                         if ( IS_WIN95 )
                            TileWindows( hWndClient, MDITILE_VERTICAL,
                                         NULL, 0, NULL );
```

```
                    else
                       SendMessage( hWndClient, WM_MDITILE,
                                    MDITILE_VERTICAL, 0 );
                    break;
        .
        .
        .
```

TRANSLATEMDISYSACCEL

WIN32S WINDOWS 95 WINDOWS NT

Description	An MDI application must call TranslateMDISysAccel() in its main message loop before calling the normal TranslateAccelerator() function.
Syntax	BOOL TranslateMDISysAccel(HWND hWndClient, LPMSG lpMsg)
Parameters	
hWndClient	HWND: The window handle of the MDI client window.
lpMsg	LPMSG: Pointer to the MSG structure containing the current message.
Returns	If the message was handled by this function, the return value is TRUE, and no further message processing is required. If this message returns FALSE, the message was not processed, and normal message processing should continue.
Include File	winuser.h
See Also	TranslateAccelerator()
Example	The following sequence of code shows the main message loop for a typical MDI application.

```
        .
        .
        .
while( GetMessage( &msg, NULL, 0, 0) )
{
   if ( hWndClient && TranslateMDISysAccel( hWndClient, &msg ) )
      continue;

   TranslateMessage( &msg );
   DispatchMessage( &msg );
}
        .
        .
        .
```

WM_MDIACTIVATE

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDIACTIVATE is a message sent to the MDI client window to force one of the MDI child windows to become the active window. This is usually done by the default window procedure handler in response to predefined accelerator keys. If an MDI child window receives this message, it indicates that the MDI child window is either becoming the active windo, or is no longer the active window.
Parameters	
wParam	HWND: The window handle of the MDI child window to be activated.
lParam	HWND: When this message is sent, this parameter is unused ans should be set to zero. When this message is received by an MDI child window, this parameter is the window handle of the MDI child window which is being deactivated.
Returns	LRESULT: If this message is processed by an MDI child window, the return value should be set to zero.
Include File	winuser.h
Related Messages	WM_MDIGETACTIVE, WM_MDINEXT, WM_NCACTIVATE

WM_MDICASCADE

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDICASCADE is a message sent to the MDI client window to cause all nonminimized MDI child windows to be rearranged in a cascaded (overlapped) arrangement.
Parameters	
wParam	UINT: The only value currently supported for this parameter is MDITILE_SKIPDISABLED, which prevents disabled MDI child windows from being moved.
lParam	LPARAM: Not used, set to zero.
Returns	BOOL: If successful, the return value is TRUE; otherwise, it is FALSE
Include File	winuser.h
See Also	CascadeWindows()
Related Messages	WM_MDIICONARRANGE, WM_MDITILE

WM_MDICREATE

WIN32S WINDOWS 95 .WINDOWS NT

Description	WM_MDICREATE is a message sent to the MDI client window to create a new MDI child window.
Parameters	
wParam	WPARAM: Not used, set to zero.
lParam	LPMDICREATESTRUCT: A pointer to an MDICREATESTRUCT structure. This structure is used by the MDI client window in the creation of the new MDI child window. See the definition of the MDICREATESTRUCT structure below.
Include File	winuser.h
Returns	If this message is successful, it returns the window handle of the new MDI child window; otherwise, it returns NULL.
See Also	CreateMDIWindow()
Related Messages	WM_CREATE, WM_MDIDESTROY

MDICREATESTRUCT Definition

```
typedef struct tagMDICREATESTRUCT
{
   LPCTSTR szClass;
   LPCTSTR szTitle;
   HANDLE  hOwner;
   int     x;
   int     y;
   int     cx;
   int     cy;
   DWORD   style;
   LPARAM  lParam;
} MDICREATESTRUCT;
```

szClass	LPCTSTR: A pointer to a null-terminated string specifying the name of the window class of the MDI child window. The class name must have been registered by a previous call to the RegisterClass() or RegisterClassEx() function.
szTitle	LPCTSTR: A pointer to a null-terminated string that represents the title of the MDI child window. Windows displays the title in the child window's title bar.
hOwner	HANDLE: The instance handle of the application creating the MDI client window.
x	int: The initial horizontal position, in client coordinates, of the MDI child window. If this member is CW_USEDEFAULT, the MDI child window is assigned the default horizontal position.
y	int: The initial vertical position, in client coordinates, of the MDI child window. If this member is CW_USEDEFAULT, the MDI child window is assigned the default vertical position.

cx	int: The initial width, in device units, of the MDI child window. If this member is CW_USEDEFAULT, the MDI child window is assigned the default width.
cy	int: The initial height, in device units, of the MDI child window. If this member is set to CW_USEDEFAULT, the MDI child window is assigned the default height.
style	DWORD: The style of the MDI child window. If the MDI client window was created with the MDIS_ALLCHILDSTYLES window style, this member can be any combination of the window styles listed in the description of the CreateWindow() function. Otherwise, this member can be one or more of the values listed in Table 27-1.
lParam	LPARAM: An application-defined 32-bit value.

WM_MDIDESTROY

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDIDESTROY is a message sent to an MDI client window to cause the indicated MDI child window to be closed.
Parameters	
wParam	HWND: Indicates which MDI child window is to be closed.
lParam	LPARAM: Not used, set to zero.
Returns	LRESULT: This message always returns zero.
Include File	winuser.h
Related Messages	WM_MDICREATE

WM_MDIGETACTIVE

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDIGETACTIVE is a message sent to the MDI client window to determine which MDI child window is currently the active window.
Parameters	This message has no parameters.
Returns	HWND: If successful, this returns the window handle of the MDI child window that is currently active. Otherwise, it returns NULL.
Include File	winuser.h
See Also	GetWindowLong()
Related Messages	WM_MDIACTIVATE

WM_MDIICONARRANGE

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDIICONARRANGE is a message sent to the MDI client window to rearrange all minimized MDI child windows. Icons representing the minimized child windows are arranged along the bottom of the MDI client window.
Parameters	This message has no parameters.
Include File	winuser.h
See Also	ArrangeIconicWindows()
Related Messages	WM_MDICASCADE, WM_MDITILE, WM_MDIRESTORE

WM_MDIMAXIMIZE

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDIMAXIMIZE is a message sent to the MDI client window to maximize an MDI child window. A maximized MDI child window occupies the entire MDI client window, appends its caption text to the MDI frame window's caption, places a RESTORE button on the far right of the MDI frame window's menu bar, and places its SYSTEM menu button on the far left of the MDI frame window's menu bar.

Parameters	
wParam	HWND: The window handle of the MDI child window to be maximized.
lParam	LPARAM: Not used, set to zero.
Returns	LRESULT: The return value is always zero.
Include File	winuser.h
Related Messages	WM_MDIRESTORE

WM_MDINEXT

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDINEXT is a message sent to the MDI client window which will activate either the next MDI child window or the previous MDI child window.
Parameters	
wParam	HWND: The window handle of an MDI child window. The MDI child window to be activated is either the next or the previous MDI child window, with relation to this window.
lParam	UINT: If this parameter is zero, the next MDI child window is activated; otherwise, the previous MDI child window is activated.
Returns	The return value is always zero.
Include File	winuser.h
Related Messages	WM_MDIACTIVATE, WM_MDIGETACTIVE

WM_MDIREFRESHMENU

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDIREFRESHMENU is a message sent to the MDI client window to refresh the MDI frame window's menu bar. This is usually done after the application has made some modifications to the menu. After sending this message, the application must call DrawMenuBar() to redraw the menu.
Parameters	This message has no parameters.
Returns	HMENU: If successful, the return value is the handle of the MDI frame windows menu; otherwise, it is NULL.
Include File	winuser.h
See Also	DrawMenuBar()
Related Messages	WM_MDISETMENU

WM_MDIRESTORE

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDIRESTORE is a message sent to the MDI client window to restore a minimized or maximized MDI child window to its original size before being minimized or maximized.
Parameters	
wParam	HWND: Indicates the MDI child window to be restored.
lParam	LPARAM: Not used, set to zero.
Returns	LRESULT: The return value is always zero.
Include File	winuser.h
Related Messages	WM_MDIMAXIMIZE

WM_MDISETMENU

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDISETMENU is a message sent to the MDI client window to associate a different menu with the application window. You can replace either the entire menu bar or just the pop-up menu item being dynamically altered by the MDI client window.
Parameters	
wParam	HMENU: The menu handle of the menu that is to replace the MDI frame window's menu bar. If you're not replacing the entire menu bar, set this parameter to NULL
lParam	HMENU: The menu handle of the menu that is to replace the MDI frame windows pop-up menu that is being dynamically altered by the MDI client window. If you're not replacing the pop-up menu, this parameter should be NULL.
Returns	HMENU: If successful, the return value is the menu handle of the MDI frame window's menu bar; otherwise, it is NULL.
Include File	winuser.h
See Also	DrawMenuBar()
Related Messages	WM_MDIREFRESHMENU

WM_MDITILE

WIN32S WINDOWS 95 WINDOWS NT

Description	WM_MDITILE is a message sent to the MDI client window to cause all nonminimized MDI child windows to be rearranged in a tiled (nonoverlapped) arrangement.
Parameters	
wParam	UINT: Specifies a tile flag. This parameter may be one of the values shown in Table 27-6.

Table 27-6. WM_MDITILE Parameter Values

Value	Description
MDITILE_HORIZONTAL	Tiles the windows so that they extend the width of the MDI client window.
MDITILE_SKIPDISABLED	Prevents disabled MDI child windows from being affected.
MDITILE_VERTICAL	Tiles the windows so that they extend the height of the MDI client window.

Returns	If successful, the return value is TRUE; otherwise, it is FALSE.
Include File	winuser.h
See Also	TileWindows()
Related Messages	WM_MDICASCADE, WM_MDIICONARRANGE