FREE eBook

LEARNING

mfc

Free unaffiliated eBook created from
Stack Overflow contributors.

#mfc

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: mfc

It is an unofficial and free mfc ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official mfc.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with mfc

Remarks

The **Microsoft Foundation Classes**, or **MFC**, is a library that provides an object-oriented wrapper around the Win32 API. By encapsulating the "raw" Win32 API in C++ classes, MFC makes it significantly easier to create GUI applications and manage resources.

MFC has been around a very long time. It was first introduced in 1992 with version 7 of Microsoft's C/C++ compiler. At this time, C++ development was just beginning to take off. Subsequent versions of Visual Studio have shipped with significantly improved versions of MFC. It is still included with the latest version of Visual Studio 2015. But its legacy roots are unfortunately quite visible. Since most of it was developed prior to the standardization of the C++ language, the MFC classes do not make full use of modern C++ features like templates, provide their own custom implementations of other standard C++ features like RTTI, and use many non-standard idioms. These facts make it nearly impossible to compile an MFC application with any compiler other than Microsoft's. On the plus side, though, MFC is well-integrated into Visual Studio, making development significantly easier.

During early development, the library was known as Application Framework Extensions (abbreviated to AFX). The marketing department later changed its name to MFC, but it was too late to change any of the code, so much of the code still references "Afx" instead of "Mfc". A noticeable example is the standard pre-compiled header file that is automatically generated by Visual Studio: it is named `StdAfx.h`.

On 7 April 2008, Microsoft released an update to the MFC classes as an out-of-band update to Visual Studio 2008 and MFC 9. The update features new user interface constructs, including the ribbons (similar to that of Microsoft Office 2007) and associated UI widgets, fully customizable toolbars, docking panes (like Visual Studio 2005) which can either be freely floated or docked to any side and document tabs. The new functionality is provided in new classes so that old applications still continue to run. This update is building on top of BCGSoft's BCGControlBar Library Professional Edition and was named as **MFC Feature Pack**.

So now MFC consists from two library with different approaches:

- Classic MFC (wrapper for Win32 API).
- MFC Feature pack (mixed set from Win32 API controls and new self-drawing controls, like Ribbon).

See also:

- MFC Desktop Applications (overview)
- MFC Reference (API reference)
- MFC Feature Pack (overview)
- MFC Feature Pack Reference (Feature Pack API reference)
- MFC Feature Pack Samples

Examples

A basic MFC program

```
// Include the MFC header:
// (you do not need to and should not include the standard Windows headers, e.g.
// Windows.h)
#include <AfxWin.h>               // MFC core and standard components
// The following header defines resource constants, such as dialog and control IDs:
#include "resource.h"

// The basic element of an MFC application is a class that inherits from CWinApp.
class CMyApp : public CWinApp
{
    // This gets called as the application gets initialized.
    virtual BOOL InitInstance()
    {
        // Initialize a CDialog object to show in a moment.
        CDialog dlg(IDD_DIALOG1);
        // Display the dialog box as a modal dialog box.
        dlg.DoModal();

        // Return FALSE from this method to exit the application.
        return FALSE;
    }
};

// The one and only application object.
CMyWinApp theApp;
```

Summary:

IDD_DIALOG1 should be the ID of a dialog box defined in a project resource file created by a resource editor, such as the one built into Visual Studio. (A resource file generally has the .rc extension.) To customize the behavior of a dialog, you can derive a new class from CDialog.

A modal dialog box runs its own message loop. The call "dlg.DoModal();" does not return until the dialog has been closed by the user.

If we had returned TRUE from InitInstance(), it would have started the application's message loop. This is used when you have a more complex, non-dialog-based app.

Read Getting started with mfc online: https://riptutorial.com/mfc/topic/2011/getting-started-with-mfc

Chapter 2: Dockable control bars (panes)

Remarks

Almost all MFC applications have toolbar and status bar - special types of the control bar that docked to top and bottom part of application main frame. But often application logic requires more the information bars docked to some side of frame, for example it may be properties bar or classes bar, preview bar, output bar and many others. Classic MFC have good solution only for toolbars and others controls that not dynamically resizable. But MFC Feature Pack have advanced docking manager that allows:

- Dynamically resize docked bars.
- Redocking to any frame sides with live preview.
- Serialize state and position of docked bars in registry.
- Docking bars to same frame side and collecting them to the tabbed pane.

All you need is to inherit your docking bar from CDockablePane class.

Examples

Docking pane to the left side of frame.

```
// Main frame class declaration
class CMainFrame
   : public CMDIFrameWndEx
{
    DECLARE_DYNAMIC(CMainFrame)

protected:
    // declare our pane
    CDockablePane m_wndPane;

    // .... other class memebers

public:
    CMainFrame();

protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    DECLARE_MESSAGE_MAP()
};


// Main frame class implementation
IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWndEx)

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWndEx)
    ON_WM_CREATE()
END_MESSAGE_MAP()

CMainFrame::CMainFrame()
```

```
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIFrameWndEx::OnCreate(lpCreateStruct) == -1)
        return -1;

    // enable frame docking to any sides
    EnableDocking(CBRS_ALIGN_ANY);

    // set the visual manager used to draw all user interface elements
    CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerWindows));

    // enable smart docking style window behavior
    CDockingManager::SetDockingMode(DT_SMART);


    // Other frame initialization code
    // ....


    // Creating the pane.
    // ID_VIEW_PANE_ID - pane ID, must be declared in resource.h
    // CRect(0, 0, 100, 100) - default pane size in floating state (pane is not docked to any
frame sides).
    // CBRS_LEFT - default side for pane docking.
    if (!m_wndPane.Create(_T("My Pane"), this, CRect(0, 0, 100, 100), TRUE, ID_VIEW_PANE_ID,
WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS | WS_CLIPCHILDREN | CBRS_LEFT | CBRS_FLOAT_MULTI)) {
        TRACE0("Failed to create Pane\n");
        return -1; // failed to create
    }
    // Enable docking and redocking pane to frame any sides (you can pass a combination of
CBRS_ALIGN_ flags)
    m_wndPane.EnableDocking(CBRS_ALIGN_ANY);

    // Dock pane to the default (left) side of the frame.
    DockPane(&m_wndPane);

    return 0;
}
```

Docking panes to the Child frame.

Some times application must have panes that docked not to the main frame, but to the child frame. Usually it's MDI application. In MFC Feature pack such child frame is inherited from `CMDIChildWndEx` class and as main frame (inherited from `CMDIFrameWndEx`) have all required code for such docking.

But there is some tricks for child frame. And this example shows them.

```
// Declare child frame
class CChildFrame : public CMDIChildWndEx
{
    DECLARE_DYNCREATE(CChildFrame)

protected:
    // declare our pane
    CDockablePane m_wndPane;
```

```
public:
    CChildFrame();

protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

    // CMDIChildWndEx class haven't serialization for state of the docking manager,
    // so we need to realize it manually.
    //
    // Docking state serialization methods:
    virtual void SaveBarState(LPCTSTR lpszProfileName) const;
    virtual void LoadBarState(LPCTSTR lpszProfileName);

protected:
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    afx_msg void OnDestroy();

    DECLARE_MESSAGE_MAP()
};



// CChildFrame Implementation

IMPLEMENT_DYNCREATE(CChildFrame, CMDIChildWndEx)

BEGIN_MESSAGE_MAP(CChildFrame, CMDIChildWndEx)
    ON_WM_CREATE()
    ON_WM_DESTROY()
END_MESSAGE_MAP()

CChildFrame::CChildFrame()
{
    // Trick#1: Add this line for enable floating toolbars
    m_bEnableFloatingBars = TRUE;
}

BOOL CChildFrame::PreCreateWindow(CREATESTRUCT& cs)
{
    if( !CMDIChildWndEx::PreCreateWindow(cs) )
        return FALSE;

    cs.style = WS_CHILD | WS_VISIBLE | WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU
        | FWS_ADDTOTITLE | WS_THICKFRAME | WS_MINIMIZEBOX | WS_MAXIMIZEBOX | WS_MAXIMIZE;

    // Trick#2: Add this line for remove the ugly client edge of the child frame.
    cs.dwExStyle &= (~WS_EX_CLIENTEDGE);

    return TRUE;
}

void CChildFrame::SaveBarState(LPCTSTR lpszProfileName) const
{
    const_cast<CChildFrame*>(this)->GetDockingManager()->SaveState(lpszProfileName);

    // Trick#3: we need to call serialization method of CMFCToolBar panes that may be docked
to the child frame.
    CObList list;
    const_cast<CChildFrame*>(this)->GetDockingManager()->GetPaneList(list, FALSE, NULL,
FALSE);
```

```
    if (list.GetCount() > 0) {
        POSITION pos = list.GetTailPosition();
        while (pos != NULL) {
            CMFCToolBar* pToolBar = DYNAMIC_DOWNCAST(CMFCToolBar, list.GetPrev(pos));
            if (pToolBar != nullptr) {
                pToolBar->SaveState(lpszProfileName);
            }
        }
    }
}

void CChildFrame::LoadBarState(LPCTSTR lpszProfileName)
{
    // Trick#3: we need to call serialization method of CMFCToolBar panes that may be docked
to the child frame.
    CObList list;
    GetDockingManager()->GetPaneList(list, FALSE, NULL, FALSE);
    if (list.GetCount() > 0) {
        POSITION pos = list.GetTailPosition();
        while (pos != NULL) {
            CMFCToolBar* pToolBar = DYNAMIC_DOWNCAST(CMFCToolBar, list.GetPrev(pos));
            if (pToolBar != nullptr) {
                pToolBar->LoadState(lpszProfileName);
            }
        }
    }

    GetDockingManager()->LoadState(lpszProfileName);
    GetDockingManager()->SetDockState();
    GetDockingManager()->ShowDelayShowMiniFrames(TRUE);

    // Trick#4: MFC BUGFIX: force assigning the child frame docking manager to all miniframes
(for details look at http://stackoverflow.com/q/39253843/987850).
    for (POSITION pos = GetDockingManager()->GetMiniFrames().GetHeadPosition(); pos != NULL;)
    {
        CWnd* pWndNext = (CWnd*)GetDockingManager()->GetMiniFrames().GetNext(pos);
        if (pWndNext != nullptr && pWndNext->IsKindOf(RUNTIME_CLASS(CPaneFrameWnd))) {
            STATIC_DOWNCAST(CPaneFrameWnd, pWndNext)->SetDockingManager(GetDockingManager());
        }
    }
}


int CChildFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    bool bRes = CMDIChildWndEx::OnCreate(lpCreateStruct) == 0;
    if (bRes)
    {
        // enable docking
        EnableDocking(CBRS_ALIGN_ANY);

        // enable Visual Studio 2005 style docking window behavior
        CDockingManager::SetDockingMode(DT_SMART);

        // Creating the pane.
        // ID_VIEW_PANE_ID - pane ID, must be declared in resource.h
        // CRect(0, 0, 100, 100) - default pane size in floating state (pane is not docked to
any frame sides).
        // CBRS_LEFT - default side for pane docking.
        if (!m_wndPane.Create(_T("My Pane"), this, CRect(0, 0, 100, 100), TRUE,
ID_VIEW_PANE_ID, WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS | WS_CLIPCHILDREN | CBRS_LEFT |
```

```
CBRS_FLOAT_MULTI)) {
            TRACE0("Failed to create Pane\n");
            return -1; // failed to create
        }
        // Enable docking and redocking pane to frame any sides (you can pass a combination of
CBRS_ALIGN_ flags)
        m_wndPane.EnableDocking(CBRS_ALIGN_ANY);

        // Dock pane to the default (left) side of the frame.
        DockPane(&m_wndPane);
    }

    // Loading dock manager state
    if (bRes) {
       LoadBarState(theApp.GetRegSectionPath(_T("ChildFrame")));
    }

    return bRes ? 0 : 1;
}

void CChildFrame::OnDestroy()
{
    // Save dock manager state
    SaveBarState(theApp.GetRegSectionPath(_T("ChildFrame")));

    CMDIChildWndEx::OnDestroy();
}
```

Read Dockable control bars (panes) online: https://riptutorial.com/mfc/topic/6706/dockable-control-bars--panes-

Chapter 3: Migrating ISAPI MFC extension (C++) VS2005 DLL project to VS2015.

Introduction

You may have seen several websites that will show how to create an ISAPI extension project but none of them will demonstrate how to migrate the existing legacy ISAPI extension (VS2005) project to VS2015. I had faced similar issue while I was working on one of such requirement. This article demonstrates the experimental waythat I had took to solve my issue.

Remarks

I was working on a migration task where I came across a project. The legacy ISAPI project was an MFC extension DLL project built in VS2005 and after googling few things I came to know that the MFC ISAPI classes (CHttpServerContext, CHttpServer etc.) are not being shipped after VS2005. Microsoft recommends using ISAPI Entry-Point functions from the Microsoft Internet Information Services (IIS) software development kit (SDK) instead of MFC ISAPI classes. I thought it might require a complete rewriting of the project. However fortunately I took an experimental way which worked in my case. I thought to write this article which might help someone to solve the similar issue.

Purpose of this topic is to demonstrate the changes required for removing dependency of MFC from MFC ISAPI Extension DLL without rewriting of complete code. Example shown above doesn't contain whole implementation, it is just for demonstration purpose.

Examples

Example:

Purpose of the project: Download particular file based on message id sent as a request from client to server. File will be created on server and transmitted to client.

Here, I haven't share the whole code but shown what is necessary as the purpose is to demonstrate the changes required for migration.

To make the changes. I took the reference of existing afxisapi.h, afxisapi.inl and HttpExt.h which are being shipped with VS2005.

- Use EXTENSION_CONTROL_BLOCK instead of CHttpServerContext. EXTENSION_CONTROL_BLOCK is a main structure which is used by IIS and the ISAPI extension to exchange the information. It contains all the information about new request.
- Remove the dependency of CHttpServer class from your MFC extension class. Remove http command parse message mapping if it is used in your project. Macros like

ON_PARSE_COMMAND, BEGIN_PARSE_MAP, END_PARSE_MAP and DEFAULT_PARSE_COMMAND were being removed after VS2005.

- Implement your own GetExtensionVersion, TerminateExtension and HttpExtensionProc function in your MFC extension class which was previously derived from CHttpSever class. These function
- GetExtensionVersion() The GetExtensionVersion function is the first entry-point function in IIS. This function allows your ISAPI extension to register its version information with IIS.

```
BOOL  CISAPIExtension::GetExtensionVersion( HSE_VERSION_INFO* pVer )
{
    ISAPIVERIFY(
       ::LoadString( AfxGetResourceHandle(),
                           IDS_SERVER,
                           sz,
                           HSE_MAX_EXT_DLL_NAME_LEN) );

       _tcscpy_s( pVer->lpszExtensionDesc, ( strlen( sz ) + 1U ), sz )
}
```

- HttpExtensionProc() : The HttpExtensionProc function is the main entry point for an ISAPI extension called by IIS. It exposes methods that IIS uses to access the functionality exposed by the extension. Extract the request information from extension control block and do your processing.

```
DWORD CISAPIExtension::HttpExtensionProc(LPEXTENSION_CONTROL_BLOCK pECB)
{
try
{
    if (strcmp(pECB->lpszMethod, "POST") == 0)
    {
        if (0 < pECB->cbTotalBytes)
        {
            unsigned long ulSize = pECB->cbTotalBytes;
            char* lpszContent = new char[ulSize + 1];
            if (NULL != lpszContent)
            {
                memcpy(lpszContent, pECB->lpbData, ulSize);
                lpszContent[ulSize] = '\0';

                std::string message_id;
                // Extract Message id from lpszContent using string operation.

                if(strcmp(pECB->lpszQueryString, "DownloadFile") == 0)
                {
                    DownloadFileFunction(pECB, message_id));
                }

                delete[] lpszContent;
                lpszContent = NULL;
            }
        }
    }
}
catch (...)
```

```
{

}
return HSE_STATUS_SUCCESS;
}
```

- Use ServerSupportFunction of EXTENSION_CONTROL_BLOCK instead of TransmitFile function of CHttepServerContext. Previously TransmitFile() were used for File Transfer.
- Use HSE_REQ_TRANSMIT_FILE in ServerSupportFunction for file transfer.
- Pull the transmit file information in HSE_TF_INFO structure.
- Create a callback function (AsyncIOCompletionFunction) that will be called on asynchronous I/O completion.

```
void WINAPI  AsyncIOCompletionFunction(
EXTENSION_CONTROL_BLOCK * pECB,
PVOID    pContext,
DWORD    cbIO,
DWORD    dwError)
{
    HSE_CUSTOM_ERROR_INFO* pErrorInfo = (HSE_CUSTOM_ERROR_INFO*)pContext;
    DWORD dwIOStatus = dwError == NO_ERROR ?
        HSE_STATUS_SUCCESS :
        HSE_STATUS_ERROR;

    if (NULL != pErrorInfo)
    {
        //
        // log HTTP status code
        //
        pECB->dwHttpStatusCode = atoi(pErrorInfo->pszStatus);

        delete pErrorInfo;
        pErrorInfo = NULL;
    }

    pECB->ServerSupportFunction(
        pECB->ConnID,
        HSE_REQ_DONE_WITH_SESSION,
        &dwIOStatus,
        (LPDWORD)NULL,
        (LPDWORD)NULL);
}
```

Code snippet for DownloadFileFunction()

```
HSE_TF_INFO pHSEInfo;

pHSEInfo.hFile            = hFile;
pHSEInfo.dwFlags          = HSE_IO_ASYNC;
pHSEInfo.pHead            = (PVOID) pvHeader;
pHSEInfo.HeadLength       = dwHeaderLen;
pHSEInfo.pContext         = 0;
pHSEInfo.pTail            = 0;
pHSEInfo.TailLength       = 0;
pHSEInfo.BytesToWrite     = 0;
pHSEInfo.Offset           = 0;
pHSEInfo.pfnHseIO         = AsyncIOCompletionFunction; // callback function that will     be
```

```
called on asynchronous I/O completion
pHSEInfo.pszStatusCode    = LPCSTR("200 OK");

bResult = m_pCtxt->ServerSupportFunction(m_pCtxt->ConnID, HSE_REQ_TRANSMIT_FILE, &pHSEInfo, 0,
0);
```

Read Migrating ISAPI MFC extension (C++) VS2005 DLL project to VS2015. online: https://riptutorial.com/mfc/topic/10026/migrating-isapi-mfc-extension--cplusplus--vs2005-dll-project-to-vs2015-

Chapter 4: Multithreading

Remarks

MFC supports worker threads and gui threads (threads with message loops). See https://msdn.microsoft.com/en-us/library/975t8ks0.aspx for more documentation.

Examples

Simple AfxBeginThread Worker Thread Example

This example shows a call of AfxBeginThread that starts the worker thread and an example worker thread procedure for that thread.

```
// example simple thread procedure.
UINT __cdecl threadProc(LPVOID rawInput)
{
    // convert it to the correct data type. It's common to pass entire structures this way.
    int* input = (int*)rawInput;
    // TODO: Add your worker code...
    MessageBox(0,"Inside thread!",0,0);
    // avoid memory leak.
    delete input;
    return 0;
}
// ...
// somewhere that gets called when you want to start the thread...
int* input = new int;
*input = 9001;
AfxBeginThread(threadProc, input);
// after this, the message box should appear, and the rest of your code should continue
// running.
```

Read Multithreading online: https://riptutorial.com/mfc/topic/6577/multithreading

Credits

S. No	Chapters	Contributors
1	Getting started with mfc	23W, aquirdturtle, Cody Gray, Community
2	Dockable control bars (panes)	23W
3	Migrating ISAPI MFC extension (C++) VS2005 DLL project to VS2015.	Rockford
4	Multithreading	aquirdturtle