Clase de baza in MFC

CObject

#include <afx.h>

CObject este clasa de baza principala pentru MFC. Majoritatea claselor din MFC sunt drivate din aceasta clasa. **CObject** furnizeaza urmatoarele servicii:

- suport de serializare;
- informatii despre clasa in timpul executiei;
- diagnosticare obiecte;
- compatibilitate cu clasele colectie (CArray, CList, etc.).

CObject nu suporta mostenirea multipla si **CObject** trebuie sa fie cel mai din stanga ierarhiei in cazul derivarii.
Daca folosim **CObject** atunci in clasele derivate putem beneficia de macro-urile:

DECLARE_DYNAMIC si **IMPLEMENT_DYNAMIC**, permite accesul in timpul executiei la numele clasei si pozitia acesteia in ierarhie.
DECLARE_SERIAL si **IMPLEMENT_SERIAL**, inclu toata functionalitatea macrourilor de mai sus, si permit unui obiect sa fie serializat (scris/citit in/din arhiva).

Exemplu pentru serializare

```
class XPersoana : public CObject {
pubic:
	// Interfata
private:
	CString		m_Nume;
	WORD		m_Varsta;
protected:
	virtual void Serialize(CArchive& ar);
};
```

Implementarea functiei **Serialization()** pentru aceasta clasa ar putea fi:

```
void XPersoana::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
		ar << m_Nume << m_Varsta;
	else
		ar >> m_Nume >> m_Varsta;
}
```

Cand avem nevoie sa memoram o instanta a clasei **XPersoana** pe disc sau sa o citim din disc, vom apela functia **Serialize()**.

Exemplu

```
class CMyDocument : public CDocument
{...
XPersoana m_persoane[100];
...
};

void CMyDocument::Serialize(CArchive& ar)
{
for (int i=0;i<100;i++)
      m_persoane[i].Serialize(ar);
}
```

Consideram in continuare ca printre aceste persoane exista una care este manager si deci are in subordine alte persoane. Vom construi clasa

```
class XManager : public CPersoana
{
      public:
            // Interface
      private:
            CList<XPersoana*, XPersoana*> m_subordonati;
};
```

unde subordonatii sunt constituiti intr-o lista simplu inlantuita.

Schimbam si implementarea clasei **`CMyDocument`** astfel:

```
class CMyDocument : public CDocument
{
      ...
      CList<XPersoana*, XPersoana*> m_persoane;
      ...
};
```

In timpul executiei, lista inlantuita ar putea arata astfel:

m_persoane -> XPersoana(“Popescu”, 20) -> XManager(“Zetu”,20) -> XPersoana(“Zoe”,12) -> etc.

Deci nu mai avem un proces simplu de serializare.
Adaugand macro-ul **DECLARE_SERIAL** in definitia clasei si **IMPLEMENT_SERIAL** in implementarea clasei, un pointer la o instanta a clasei poate fi memorat si realocat dintr-o arhiva.

In concluzie implementarea completa pentru aceasta clasa este:

```
class XPersoana : public CObject
{
public:
      // Interfata
```

```
private:
	CString m_Nume;
	WORD   m_Varsta;
protected:
	virtual void Serialize(CArchive& ar);
	DECLARE_SERIAL(XPersoana)
};

class XManager : public XPersoana
{
public:
	// Interfata
private:
	CList<XPersoana*, XPersoana*> m_subordonati;
protected:
	void Serialize(CArchive& ar);
	DECLARE_SERIAL(XManager)
};

IMPLEMENT_SERIAL(XPersoana, CObject, 1)
IMPLEMENT_SERIAL(XManager,  XPersoana, 1)

//
// Aceasta este o functie helper pentru clasa colectie
// template CList si ne spune cum memoreaza obiecte
// de tipul XPersoana*
//

void SerializeElements(CArchive& ar, XPersoana** pElemente, int
nCount)
{
	for (int i=0;i < nCount; i++) {
		if (ar.IsStoring())
			ar << pElemente[i];
		else
			ar >> pElemente[i];
	}
}
void XPersoana::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
		ar << m_Nume << m_Varsta;
	else
		ar >> m_Nume >> m_Varsta;
}
void XManager::Serialize(CArchive& ar)
{
	XPersoana::Serialize(ar);
	m_subordonati.Serialize(ar);
}
```

```
void CMyDocument::Serialize(CArchive& ar)
{
	m_persoane.Serialize(ar);
}
```

CCmdTarget

CCmdTarget este clasa de bază pentru arhitectura de tratare a mesajelor din biblioteca MFC. Dacă se doreşte crearea unei noi clase ce trebuie să trateze mesaje, aceasta trebuie derivată din **CCmdTarget** sau dintr-un descendent al acesteia. Metoda **OnCmdMsg ( )** este folosită pentru rutarea, distribuirea mesajelor şi tratarea acestora.
În plus clasa CCmdTarget mai gestionează trecerea cursorului în starea de aşteptare (cursor cu clepsidră) şi ieşirea din această stare folosind metodele **BeginWaitCursor ( ), EndWaitCursor ( )** şi **RestoreWaitCursor ( ).**

Clase derivate din **CCmdTarget: CView, CWinApp, CDocument, CWnd** si **CFrameWnd**.

Pentru a lucra cu comenzi va trebui sa completam harta de mesaje (se face corespondenta intre mesaj si functia ce-l trateaza) iar in majoritatea cazurilor acest lucru este facut de ClassWizard. Bine-nteles codul din functia ce trateaza mesajul trebuie scris de noi.

In general mesajele sunt trimise ferestrei cadru principale, dar comenzile sunt rutate apoi catre alte obiecte.

In mod normal o clasa ruteaza comenzile la alte obiecte pentru a le da sansa sa trateze comanda. Daca comanda nu este trata de nici un obiect atunci se cauta in harta de mesaje a clasei pentru a vedea daca mesajul are asociata o functie de tratare. In situatia cind clasa nu trateaza comanda, aceasta este rutata catre clasa de baza a clasei curente. Vom explica pe larg aceasta rutare intr-un curs viitor.

Urmatorul exemplu este din MSDN. Se explica sintaxa metodei **OnCmdMsg** si apoi se da un exemplu.

CCmdTarget::OnCmdMsg

virtual BOOL OnCmdMsg(UINT *nID*, **int** *nCode*, **void*** *pExtra*, **AFX_CMDHANDLERINFO*** *pHandlerInfo* **);**

Return Value
Nonzero if the message is handled; otherwise 0.

Parameters

nID	Contains the command ID.
nCode	Identifies the command notification code.
pExtra	Used according to the value of *nCode*.
pHandlerInfo	If not **NULL**, **OnCmdMsg** fills in the **pTarget** and **pmf** members of the *pHandlerInfo* structure instead of dispatching the command. Typically, this parameter should be **NULL**.

Remarks

Called by the framework to route and dispatch command messages and to handle the update of command user-interface objects. This is the main implementation routine of the framework command architecture.

At run time, **OnCmdMsg** dispatches a command to other objects or handles the command itself by calling the root class **CCmdTarget::OnCmdMsg**, which does the actual message-map lookup.

Example

```
// This example illustrates extending the framework's standard command
// route from the view to objects managed by the view.  This example
// is from an object-oriented drawing application, similar to the
// DRAWCLI sample application, which draws and edits "shapes".

BOOL CMyView::OnCmdMsg(UINT nID, int nCode, void* pExtra,
      AFX_CMDHANDLERINFO* pHandlerInfo)
{
   // Extend the framework's command route from the view to
   // the application-specific CMyShape that is currently selected
   // in the view. m_pActiveShape is NULL if no shape object
   // is currently selected in the view.

if ((m_pActiveShape != NULL)
      && m_pActiveShape->OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))
  return TRUE;

   // If the object(s) in the extended command route don't handle
   // the command, then let the base class OnCmdMsg handle it.
   return CView::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);
}

// The command handler for ID_SHAPE_COLOR (menu command to change
// the color of the currently selected shape) was added to
// the message map of CMyShape (note, not CMyView) using ClassWizard.

// The menu item will be automatically enabled or disabled, depending
// on whether a CMyShape is currently selected in the view, that is,
// depending on whether CMyView::m_pActiveView is NULL.  It is not
// necessary to implement an ON_UPDATE_COMMAND_UI handler to enable
// or disable the menu item.

BEGIN_MESSAGE_MAP(CMyShape, CCmdTarget)
   //{{AFX_MSG_MAP(CMyShape)
   ON_COMMAND(ID_SHAPE_COLOR, OnShapeColor)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Asemanator hartilor de mesaje exista harti ce expun automatizarea prin implementarea interfetei IDispatch.

CWinThread

Un obiect din clasa **CWinThread** reprezintă un fir de execuţie dintr-o aplicaţie. Firul de execuţie principal este un obiect al clasei **CWinApp** ce este o clasă derivată din **CWinThread**. Pe lângă firul de execuţie principal se mai pot folosi şi alte fire de execuţie, folosind obiecte **CWinThread**.

Exista doua tipuri de fire de executie suportate de **CWinThread:**

1. **fire de lucru** (fara interfata utilizator, deci nu au bucla de mesage);
2. **fire cu interfata utilizator**.

O problema importanta legata de firele de executie o constituie sincronizarea acestora (executie sincronizata).

Membrii clasei CWinThread (selectie din MSDN)

Date

m_bAutoDelete	Specifies whether to destroy the object at thread termination.
m_hThread	Handle to the current thread.
m_nThreadID	ID of the current thread.
m_pMainWnd	Holds a pointer to the application's main window.
m_pActiveWnd	Pointer to the main window of the container application when an OLE server is in-place active.

Operations

GetMainWnd	Retrieves a pointer to the main window for the thread.
GetThreadPriority	Gets the priority of the current thread.
PostThreadMessage	Posts a message to another **CWinThread** object.
ResumeThread	Decrements a thread's suspend count.
SetThreadPriority	Sets the priority of the current thread.
SuspendThread	Increments a thread's suspend count.

Overridables

ExitInstance	Override to clean up when your thread terminates.
InitInstance	Override to perform thread instance initialization.
OnIdle	Override to perform thread-specific idle-time processing.
PreTranslateMessage	Filters messages before they are dispatched to the Windows functions TranslateMessage and DispatchMessage.
IsIdleMessage	Checks for special messages.
Run	Controlling function for threads with a message pump. Override to customize the default message loop.

CWinApp

Derivata din CWinThread.

Clasa **CWinApp** este clasa de bază pentru obiectul aplicaţie. Clasa aplicaţie din MFC încapsulează iniţializarea, execuţia şi terminarea unei aplicaţii Windows.

Fiecare aplicaţie MFC poate conţine doar un singur obiect derivat din CWinApp. Acest obiect este global şi este construit înaintea construirii ferestrelor, fiind disponibil atunci când sistemul de operare Windows apelează funcţia WinMain, care este oferită de biblioteca MFC.

Când se derivează o clasă aplicaţie din **CWinApp**, se suprascrie funcţia membru **InitInstance ( )** pentru a se construi şi iniţializa noua aplicaţie.

Pe lângă funcţiile membru ale lui **CWinApp**, MFC oferă funcţii globale pentru a obţine informaţii despre obiectul aplicaţie curent:

- **AfxGetApp ( )** – returnează un pointer la obiectul aplicaţie curent;
- **AfxGetInstanceHandle ( )** – handle la instanţa aplicaţie curentă;
- **AfxGetResourceHandle ( )** – handle la resursele aplicaţiei;
- **AfxGetAppName ( )** – numele aplicaţiei.

Exemplu

```
class Cmfc1App : public CWinApp
{
public:
	Cmfc1App();


// Overrides
public:
	virtual BOOL InitInstance();

// Implementation
	afx_msg void OnAppAbout();
	DECLARE_MESSAGE_MAP()
};
```

iar in implementare

```
BOOL Cmfc1App::InitInstance()
{
// InitCommonControls() is required on Windows XP if an application
// manifest specifies use of ComCtl32.dll version 6 or later to enable
// visual styles.  Otherwise, any window creation will fail.
	InitCommonControls();

	CWinApp::InitInstance();

	// Initialize OLE libraries
	if (!AfxOleInit())
	{
		AfxMessageBox(IDP_OLE_INIT_FAILED);
		return FALSE;
	}
	AfxEnableControlContainer();
```

```
// Standard initialization
// If you are not using these features and wish to reduce the size
// of your final executable, you should remove from the following
// the specific initialization routines you do not need
// Change the registry key under which our settings are stored
// TODO: You should modify this string to be something appropriate
// such as the name of your company or organization

      SetRegistryKey(_T("Local AppWizard-Generated Applications"));
      LoadStdProfileSettings(4);
// Load standard INI file options (including MRU)

// Register the application's document templates.  Document templates
//  serve as the connection between documents, frame windows and views
      CSingleDocTemplate* pDocTemplate;
      pDocTemplate = new CSingleDocTemplate(
            IDR_MAINFRAME,
            RUNTIME_CLASS(Cmfc1Doc),
            RUNTIME_CLASS(CMainFrame),       // main SDI frame window
            RUNTIME_CLASS(Cmfc1View));
      if (!pDocTemplate)
            return FALSE;
      AddDocTemplate(pDocTemplate);
      // Parse command line for standard shell commands, DDE, file open
      CCommandLineInfo cmdInfo;
      ParseCommandLine(cmdInfo);

// Dispatch commands specified on the command line.  Will return FALSE if
// app was launched with /RegServer, /Register, /Unregserver or /Unregister.

      if (!ProcessShellCommand(cmdInfo))
            return FALSE;
// The one and only window has been initialized, so show and update it
      m_pMainWnd->ShowWindow(SW_SHOW);
      m_pMainWnd->UpdateWindow();

// call DragAcceptFiles only if there's a suffix
//  In an SDI app, this should occur after ProcessShellCommand

      return TRUE;
}
```

CWnd

Clasa CWnd este clasa de bază pentru toate celelalte **clase fereastră** oferite de MFC. Această clasă are o mare parte din funcţiile necesare în obiectele frame, view, controale, etc.
Obiectul fereastră este un obiect al clasei **CWnd** sau derivat din clasa **CWnd** care este creat direct de către program. Fereastra, pe de altă parte este un handle către o structură internă din Windows care conţine resursele ferestrei respective.

CFrameWnd

Clasa **CFrameWnd** conţine o implementare implicită pentru următoarele funcţii ale ferestrei principale dintr-o aplicaţie Windows:

- **menţine ordinea cu ferestrele de vizualizare active;**
- **comenzile şi mesajele de notificare le trimite ferestrei de vizualizare active;**
- **modificarea textului ce apare pe bara de titlu a ferestrei în funcţie de fereastra de vizualizare activă;**
- **se ocupă de poziţionarea barelor de control, a ferestrelor de vizualizare şi a altor ferestre copil, în zona client a ferestrei principale;**
- **bara de meniu;**
- **meniul sistem al aplicaţiei;**
- **acceleratori (tratarea tastelor speciale);**
- **starea iniţială a aplicaţiei (minimizată, maximizată);**
- **help senzitiv la context;**
- **bara de stare;**
- **se ocupă de perechile document-view.**

Exemplu

```
class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
	CMainFrame();
	DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
public:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
```

```
// Implementation
public:
	virtual ~CMainFrame();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
	CStatusBar  m_wndStatusBar;
	CToolBar    m_wndToolBar;

// Generated message map functions
protected:
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	DECLARE_MESSAGE_MAP()
};
```

iar in implementare avem (cod partial pentru crearea ferestrei)

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

	if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD |
		WS_VISIBLE | CBRS_TOP
		| CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
		CBRS_SIZE_DYNAMIC) ||
		!m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
	{
		TRACE0("Failed to create toolbar\n");
		return -1;      // fail to create
	}

	if (!m_wndStatusBar.Create(this) ||
		!m_wndStatusBar.SetIndicators(indicators,
		  sizeof(indicators)/sizeof(UINT)))
	{
		TRACE0("Failed to create status bar\n");
		return -1;      // fail to create
	}

// TODO: Delete these three lines if you don't want
// the toolbar to be dockable
	m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
	EnableDocking(CBRS_ALIGN_ANY);
	DockControlBar(&m_wndToolBar);

	return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
```

```
      if( !CFrameWnd::PreCreateWindow(cs) )
            return FALSE;
      // TODO: Modify the Window class or styles here by modifying
      //  the CREATESTRUCT cs

      return TRUE;
}
```

Structura CREATESTRUCT este asemanatoare cu structura WNDCLASSEX.

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

Document templates - Sabloane de document

Aplicaţiile MFC folosesc implicit un model de programare ce separă datele programului de partea de afişare a acestor date şi de majoritatea interacţiunilor dintre utilizator şi date. În acest model, un obiect document citeşte şi scrie datele pe disc şi mai poate oferi nişte funcţii de lucru cu aceste date.
Un obiect distinct se ocupă de partea de vizualizare a datelor într-o anumită fereastră şi tratează interacţiunea utilizatorului cu acestea.
Obiectul de vizualizare poate citi datele din document şi le poate modifica la acţiunea utilizatorului.

Modelul document/view este aplicabil şi în situaţia în care există mai multe ferestre de vizualizare pentru un document, lăsând libertatea fiecărui obiect de vizualizare să-şi afişeze datele, în timp ce partea de cod comună tuturor ferestrelor de vizualizare (cum ar fi partea de calcule) se poate implementa în document.

Documentul se mai ocupă şi de reactualizarea ferestrelor de vizualizare dacă datele au fost modificate (de către document sau de către una din ferestrele de vizualizare).
Arhitectura MFC document/view suportă o implementare uşoară a ferestrelor de vizualizare multiple, tipuri de documente diferite, ferestre împărţite (splitter windows), şi alte tipuri de caracteristici ale interfeţelor. La baza modelului document/view stau următoarele patru clase:

- **CDocument (sau COleDocument);**
- **CView;**
- **CFrameWnd;**
- **CDocTemplate.**

Părţile din MFC cele mai vizibile, atât pentru utilizator cât şi pentru programator, sunt documentul şi ferestrele de vizualizare. Cea mai mare parte din munca investită într-un proiect constă în scrierea claselor document şi view.

Clasa **CDocument** oferă funcţiile de bază pentru clasele document specifice aplicaţiei. Un document reprezintă un bloc de date pe care utilizatorul îl poate deschide cu comanda Open, salva cu comanda Save, etc.

Clasa **CView** stă la baza claselor view ale aplicaţiei. Un view este ataşat unui document şi funcţionează ca un intermediar între document şi utilizator: view-ul construieşte în fereastră o imagine a documentului şi interpretează acţiunile utilizatorului şi le transmite documentului.
În figura următoare este prezentată relaţia dintre document şi view:

Documentele, ferestrele de vizualizare asociate şi ferestrele cadru care conţin ferestrele de vizualizare pentru un document sunt create de un template document. Acesta este responsabil de crearea şi gestionarea tuturor documentelor de acelaşi tip.
Orice aplicaţie MFC SDI creată cu AppWizard are cel puţin un document template. Acest template crează şi defineşte relaţiile dintre document, fereastra cadru şi fereastra de vizualizare. Când este creat un nou element sau când este deschis un document, acest template este folosit pentru a crea cele trei elemente în următoarea ordine: documentul, fereastra cadru şi fereastra de vizualizare. Pentru o aplicaţie MDI mai apare un pas în plus faţă de aplicaţia SDI: crearea ferestrei cadru principale a aplicaţiei înaintea creării documentului.

Template-ul document mapează documentul, fereastra cadru şi fereastra de vizualizare pe clasele proprii aplicaţiei.
Crearea unui template (creat implicit de AppWizard):

```
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
            IDR_MAINFRAME,
            RUNTIME_CLASS(CSdiDoc),
            RUNTIME_CLASS(CMainFrame),
      // main SDI frame window
            RUNTIME_CLASS(CSdiView));
AddDocTemplate(pDocTemplate);
```

Secvenţa de apeluri la crearea documentului:

- **CMyApp::InitInstance ( );**
- **dacă aplicaţia este MDI atunci se crează şi afişează fereastra cadru principală;**
- **parcurge şi procesează linia de comandă (dacă nu există se apelează OnFileNew ( ));**
- **se selectează template-ul document;**
- **se crează un document gol prin CDocTemplate::OpenDocumentFile ( );**
- **CDocTemplate::CreateNewDoc ( );**
- **constructorul pentru CMyDocument;**
- **CMyDocument::OnNewDocument ( ).**

Crearea ferestrei cadru:

- **CDocTemplate::CreateNewFrame ( )**
- **constructorul pentru CMainFrame;**
- **CMainFrame::LoadFrame ( );**
- **CMainFrame::PreCreateWindow ( );**
- **CMainFrame::OnCreate ( );**
- **CMainFrame::OnCreateClient ( );**

Crearea ferestrei de vizualizare:

- **CMainFrame::CreateView ( );**
- **constructorul pentru CMyView;**
- **CMyView::PreCreateWindow ( );**
- **CMainFrame::InitialUpdateFrame ( );**
- **CMyView::OnInitialUpdate ( );**
- **CMyView::OnActivateFrame ( );**
- **CMainFrame::ActivateFrame ( );**
- **CMyView::OnActivateView ( );**

CDocument

Clasa **CDocument** furnizeaza functionalitatea de baza in arhitectura document-view implementata in MFC si are ca scop gestionarea documentului aplicatiei.
CDocument suporta operatii standard de creare, incarcare si salvare a documentului.
O aplicatie poate suporta mai mult decat un document. Fiecare tip are asociat un document template (sablon). Acest sablon specifica resursele utilizate pentru acel tip de document.
Utilizatorul interactioneaza cu un document prin intermediul unui obiect CView. Un obiect CView reprezinta o vizualizare a documentului in zona client.

Un document poate avea asociate mai multe vizualizari.
Un document primeste comenzi forward-ate de vizualizarea activa precum si comenzi din meniu (Open, Save).

CView

Clasa **CView** (obiecte instantiate direct sau indirect) furnizeaza vizualizarea documentului.
O vedere actioneaza ca un intermediar intre document si utilizator.
O vedere este o fereastra descendenta din fereastra cadru.

O vedere este responsabila de tratarea mai multor tipuri de intrari: tastatura, mouse, operatii de drag & drop, comenzi din meniu, toobar sau bare de defilare (scroll bars).

Metoda cea mai importanta din aceasta clasa este **OnDraw**, responsabila pentru desenarea in zona client.
O alta metoda folosita este cea care face legatura intre document si vizualizare: **GetDocument(),** functie ce returneaza un pointer la obiectul de tip document.

Alte clase de vizuallizare: CCtrlView, CDaoRecordView, CEditView, CFormView, CListView, CRecordView, CRichEditView, CScrollView, CTreeView.

Harta de mesaje

Hărțile de mesaje sunt părți ale modelului MFC de programare Windows. În loc de a scrie funcția **WinMain()** care trimite mesaje la procedura fereastră (funcția) WindProc() şi apoi să controlăm ce mesaj a fost trimis pentru a activa funcția corespunzătoare, vom scrie doar funcția care tratează mesajul şi vom adăuga mesajul la harta de mesaje a clasei. Cadrul de lucru va face operațiunile necesare pentru a ruta acest mesaj în mod corect.

Construirea hărții de mesaje

Hărțile de mesaje se construiesc în două etape. **Declararea** hărtii de mesaje (macro **DECLARE_MESSAGE_MAP()**) se face în fişierul .h al clasei, iar **implementarea** se face in fişierul .cpp al clasei **(BEGIN_MESSAGE_MAP() ... END_MESSAGE_MAP()).**

Exemplu

```
//{{AFX_MSG(CShowStringApp)
     afx_msg void OnAppAbout();
 //the ClassWizard will add and remove member functions here.
 // DO NOT EDIT what you see in these blocks of generated code !
     //}}AFX_MSG
     DECLARE_MESSAGE_MAP()
```

Se declară funcția **OnAppAbout()** care este prefixată cu **afx_msg** ce constituie un comentariu pentru compilatorul de C++, dar care indică vizual că această funcție tratează un mesaj. Această funcție o vom găsi şi în cadrul macro-ului **BEGIN_MESSAGE_MAP(),** ca un parametru al macro-ului **ON_COMMAND().** Primul parametru al acestui din urmă macro este ID-ul mesajului (comenzii în acest caz), iar al doilea numele funcției ce tratează acest mesaj.

Cod in .cpp

```
BEGIN_MESSAGE_MAP(CShowStringApp, CWinApp)
     //{{AFX_MSG_MAP(CShowStringApp)
     ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
//DO NOT EDIT what you see in these blocks of generated code!
     //}}AFX_MSG_MAP
     // Standard file based document commands
     ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
     ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
     ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()
```

Macro-ul **DECLARE_MESSAGE_MAP()** adaugă date mebru şi funcţii la clasa respectivă. Practic se declară o tabelă cu un număr de intrări variabil (sfârşitul tabelei este marcat (completat) de **END_MESSAGE_MAP()**) şi funcţii care operează cu acest elementele acestui tabel.

Macro-uri pentru harta de mesaje

BEGIN_MESSAGE_MAP şi END_MESSAGE_MAP sunt macro-uri care ca şi macro-ul DECLARE_MESSAGE_MAP din fişierul .h, declară anumite variabile membru şi funcţii pe care cadrul de lucru le va utiliza pentru a naviga prin hărţile de mesaje ale tuturor obiectelor din sistem Printre macro-urile folosite în hărţile de mesaje, enumerăm:

- **DECLARE_MESSAGE_MAP**—folosit în fişierul .h pentru a declara că va exista o hartăesaje in .cpp
- **BEGIN_MESSAGE_MAP**—Marchează începutul hărţii de mesaje în fişierul sursă.
- **END_MESSAGE_MAP**—Marchează sfârşitul hărţii de mesaje în fişierul sursă.
- **ON_COMMAND**—Folosit pentru a face legătura între comenzi şi funcţiile care tratează aceste comenzi.
- **ON_COMMAND_RANGE**—Folosit pentru a face legătura între un grup de comenzi şi funcţia care le tratează.
- **ON_CONTROL**—Folosit pentru a face legătura între un mesaj de notificare al unui control şi funcţia ce-l tratează.
- **ON_CONTROL_RANGE**—Folosit pentru a face legătura între un grup de mesaje de notificare al unui control şi funcţia corespunzătoare.
- **ON_MESSAGE**—Folosit pentru a realiza legătura între un mesaj definit de utilizator şi funcţia care-l tratează.
- **ON_REGISTERED_MESSAGE**—Folosit pentru a realiza legătura între un mesaj defint de utilizator, dar înregistrat şi funcţia care-l tratează.
- **ON_UPDATE_COMMAND_UI**—Folosit pentru a indica funcţia care va face actualizarea pentru o comandă specifică.
- **ON_COMMAND_UPDATE_UI_RANGE**—Ca mai sus, dar pentru un grup de comenzi.
- **ON_NOTIFY**—Folosit pentru a indica funcţia ce va adăuga informaţii suplimentare, pentru un mesaj de notificare al unui control.
- **ON_NOTIFY_RANGE**—Ca mai sus, dar pentru un grup de mesaje de notificare al unui control. ON_NOTIFY_EX—Ca la ON_NOTIFY, dar funcţia va întoarce TRUE sau FALSE pentru a indica dacă notificarea poate fi trecută altui obiect pentru tratări suplimentare.
- ON_NOTIFY_EX_RANGE—Ca mai sus, dar se referă la un grup de comenzi de notificare.

În plus la ceste macro-uri, există peste 100 de macro-uri, unul pentru fiecare din cele mai comune mesaje. De exemplu macro-ul **ON_CREATE** pentru mesajul **WM_CREATE**, etc. În mod obişnuit aceste macro-uri sunt adăugate la clasă de către Class Wizard.

Cum lucrează harta de mesaje

Fiecare aplicaţie are un obiect moştenit din clasa **`CWinApp`** şi o funcţie membru **`Run()`.** Această funcţie apelează funcţia **`CWinThread::Run()`,** care apelează **`GetMessage()`, `TranslateMessage()`** şi **`DispatchMessage()`.**

Funcţia fereastră (în SDK) ştie handler-ul ferestrei pentru care este trimis mesajul. Fiecare obiect fereastră foloseşte acelaşi stil al clasei Windows şi aceeaşi funcţie WindProc, numită `AfxWndProc()`. MFC menţine ceva asemănător, numit *handle map*, o tabelă cu handler-ii ferestrelor şi pointeri la obiecte, şi framework-ul foloseşte aceasta pentru a trimite un pointer la obiectul C++, un `CWnd*`. În continuare el apelează `WindowProc()`, o funcţie virtuală a acestui obiect. Datorită polimorfismului, indiferent că este vorba de un Button sau o vizualizare se va apela funcţia corectă.
`WindowProc()` apelează `OnCmdMsg()`, funcţia C++ care efectiv manipulează mesajul. Mai întâi caută dacă acesta este un **mesaj, o comandă sau o notificare**. Presupunând că este un mesaj. caută în harta de mesage a clasei, folosind funcţiile şi variabilele membru adăugate la clasă de DECLARE_MESSAGE_MAP, BEGIN_MESSAGE_MAP şi END_MESSAGE_MAP. Modul de organizare al acestei tabele permite căutarea mesajului, dacă este nevoie, în toată arborescenţa clasei.

`AfxWndProc()->WindowProc()->OnCmdMsg()`

Recunoaşterea mesajelor

Există aproximativ 900 mesaje Windows.

Prefixele mesajului Windows şi Tipuri fereastră

Prefix	Window Type
ABM, ABN	Appbar
ACM, ACN	Animation control
BM, BN	Button
CB, CBN	Combo box
CDM, CDN	Common dialog box
CPL	Control Panel application
DBT	Any application (device change message)
DL	Drag list box
DM	Dialog box
EM, EN	Edit box
FM, FMEVENT	File Manager
HDM, HDN	Header control
HKM	HotKey control
IMC, IMN	IME window
LB, LBN	List box
LVM, LVN	List view
NM	Any parent window (notification message)
PBM	Progress bar
PBT	Any application (battery power broadcast)
PSM, PSN	Property sheet
SB	Status bar
SBM	Scrollbar

STM, STN	Static control
TB, TBN	Toolbar
TBM	Track bar
TCM, TCN	Tab control
TTM, TTN	ToolTip
TVM, TVN	Tree view
UDM	Up Down control
WM	Generic window

Care e diferența între mesajele care se termină în M şi cele care se termină în N? Primul este un mesaj la control (am apăsat butonul, de exemplu), al doilea este un mesaj de notificare de la control la fereastra proprietară a controlului, care are semnificația de "am fost apăsat", "s-a întâmplat ceva în control...".
Există şi mesaje care nu se termină în M (CB_) dar acționează la fel.

Înțelegerea comenzilor

O **comandă** este un tip special de mesaj. Windows generează comenzi când utilizatorul alege un articol de meniu, apasă un buton, sau altfel spune sistemului să facă ceva. Pentru un articol de meniu se primeşte mesajul **WM_COMMAND** iar pentru notificarea unui control **WM_NOTIFY**, cum ar fi selectarea dintr-un list box.

Comenzile şi notificările sunt trecute prin SO ca orice alt mesaj, până când acestea ajung la **`OnWndMsg().`** În acest punct pasarea mesajului windows încetează şi se startează rutarea comenzilor în MFC.
Mesajele de comandă au ca prim parametru, ID-ul articolului din meniu care a fost selectat sau a butonului care a fost apăsat.
Rutarea comenzilor este mecanismul pe care **OnWndMsg()** îl foloseşte pentru a trimite comanda (sau notificarea) la obiectele care pot trata acest mesaj. **Numai obiectele care sunt moştenite din CWnd pot primi mesaje**, **dar toate obiectele care sunt moştenite din CCmdTarget, incluzând CWnd şi CDocument, pot primi comenzi sau notificări**. Aceasta însemană că o clasă moştenită din `CDocument` poate avea o hartă de mesaje. Pot să nu existe mesaje în această hartă ci numai pentru comenzi şi notificări, dar tot hartă de mesaje se numeşte.

Comenzile şi notificările ajung la clasă prin mecanismul de rutare al comenzilor. `OnWndMsg()` apelează `CWnd::OnCommand()` sau `CWnd::OnNotify().` `OnCommand()` apelează `OnCmdMsg().` `OnNotify()` apelează de asemenea `OnCmdMsg().` Binențeles că ambele funcții efectuează anumite controale înainte de a face aceste apeluri.

`OnCmdMsg()` este virtuală, ceea ce înseamnă că diferite comenzi au implementări diferite. Implementarea pentru fereastra cadru trimite comanda vizualizărilor şi documentelor pe care le conține.

Comanda pentru actualizări

Folosit în special pentru actualizarea articolelor din meniu. De exemplu când se selectează text in vizualizare şi opţiunile de Copy, Cut, Paste, Undo sunt implementate aceste articole de menu vor fi afişate în starea enable sau disable (funcţie de logica programului).
Există două posibilităţi de a face acest lucru: *continuous update approach* şi *update-on-demand approach*.
Continuous update approach presupune existenţa unei tabele mari ce conţine câte o intrare pentru fiecare meniu şi un flag care va indica dacă opţiunea este disponibilă sau nu.
Cea de-a doua posibilitate presupune controlarea tuturor condiţiilor asupra unui articol de meniu înainte ca meniul să fie afişat. În acest caz obiectul care are meniul va şti mai multe despre acesta, în schimb nu toată aplicaţia va avea acces la aceste informaţii.
Tehinca MFC-ului este de a utiliza un obiect numit `CCmdUI`, o comandă a interfeţei utilizatorului, şi de a da acest obiect când se trimite mesajul CN_UPDATE_COMMAND_UI. În harta de mesaje va apărea macro-ul ON_UPDATE_COMMAND_UI.

Ce se întâmplă in realitate?

SO trimite mesajul WM_INITMENUPOPUP; clasa *`CFrameWnd`* va construi un obiect *`CCmdUI`*, setează variabilele membru ce corespund primului articol din meniu şi apelează funcţia membru `DoUpdate()`. `DoUpdate()` trimite mesajul CN_COMMAND_UPDATE_UI la ea însăşi, cu un pointer la obiectul `CCmdUI`. Se vor seta variabilele membru ale obiectului `CCmdUI` cu articolul doi din meniu şi procesul continuă până când este parcurs tot meniul. Obiectul CCmdUI este folosit pentru a valida (enable) sau invalida (disable) [*gray(disable)* sau *ungray(enable)*] articolele din meniu sau butoane.

`CCmdUI` are următoarele funcţii membru:

- `Enable()` — Are un parametru care poate lua valorile TRUE sau FALSE (implicit TRUE).
- `SetCheck()` — Marchează sau demarchează articolul.
- `SetRadio()` – Setează sau nu unul din butoanele radio al unui grup.
- `SetText()`—Setează textul unui meniu sau buton.
- `DoUpdate()`—Generează mesajul.

`Exemplu:`

```
BEGIN_MESSAGE_MAP(CWhoisView, CFormView)
      ...
      ON_UPDATE_COMMAND_UI(ID_EDIT_PASTE, OnUpdateEditPaste)
      ...
END_MESSAGE_MAP()

void CWhoisView::OnUpdateEditPaste(CCmdUI* pCmdUI)
{
 pCmdUI->Enable(::IsClipboardFormatAvailable(CF_TEXT));
}
```

Exemplu complet de aplicatie (SDI) generata de AppWizard

```
// mfc1.h : main header file for the mfc1 application
//
#pragma once

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols


// Cmfc1App:
// See mfc1.cpp for the implementation of this class
//

class Cmfc1App : public CWinApp
{
public:
	Cmfc1App();


// Overrides
public:
	virtual BOOL InitInstance();

// Implementation
	afx_msg void OnAppAbout();
	DECLARE_MESSAGE_MAP()
};

extern Cmfc1App theApp;

// MainFrm.h : interface of the CMainFrame class
//

#pragma once
class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
	CMainFrame();
	DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
public:
	virtual BOOL PreCreateWindow(CREATESTRUCT& cs);

// Implementation
public:
	virtual ~CMainFrame();
#ifdef _DEBUG
```

```
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
	CStatusBar  m_wndStatusBar;
	CToolBar    m_wndToolBar;

// Generated message map functions
protected:
	afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
	DECLARE_MESSAGE_MAP()
};


// mfc1Doc.h : interface of the Cmfc1Doc class
//


#pragma once

class Cmfc1Doc : public CDocument
{
protected: // create from serialization only
	Cmfc1Doc();
	DECLARE_DYNCREATE(Cmfc1Doc)

// Attributes
public:

// Operations
public:

// Overrides
	public:
	virtual BOOL OnNewDocument();
	virtual void Serialize(CArchive& ar);

// Implementation
public:
	virtual ~Cmfc1Doc();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
	DECLARE_MESSAGE_MAP()
};


// mfc1View.h : interface of the Cmfc1View class
//


#pragma once
```

```
class Cmfc1View : public CView
{
protected: // create from serialization only
	Cmfc1View();
	DECLARE_DYNCREATE(Cmfc1View)

// Attributes
public:
	Cmfc1Doc* GetDocument() const;

// Operations
public:

// Overrides
	public:
	virtual void OnDraw(CDC* pDC);  // overridden to draw this view
virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
	virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
	virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
	virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

// Implementation
public:
	virtual ~Cmfc1View();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
	DECLARE_MESSAGE_MAP()
public:
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
};

#ifndef _DEBUG  // debug version in mfc1View.cpp
inline Cmfc1Doc* Cmfc1View::GetDocument() const
   { return reinterpret_cast<Cmfc1Doc*>(m_pDocument); }
#endif

// mfc1.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "mfc1.h"
#include "MainFrm.h"

#include "mfc1Doc.h"
#include "mfc1View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
```

```
// Cmfc1App

BEGIN_MESSAGE_MAP(Cmfc1App, CWinApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()


// Cmfc1App construction

Cmfc1App::Cmfc1App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}


// The one and only Cmfc1App object

Cmfc1App theApp;

// Cmfc1App initialization

BOOL Cmfc1App::InitInstance()
{
	// InitCommonControls() is required on Windows XP if an application
	// manifest specifies use of ComCtl32.dll version 6 or later to enable
	// visual styles.  Otherwise, any window creation will fail.
	InitCommonControls();

	CWinApp::InitInstance();

	// Initialize OLE libraries
	if (!AfxOleInit())
	{
		AfxMessageBox(IDP_OLE_INIT_FAILED);
		return FALSE;
	}
	AfxEnableControlContainer();
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	// of your final executable, you should remove from the following
	// the specific initialization routines you do not need
	// Change the registry key under which our settings are stored
	// TODO: You should modify this string to be something appropriate
	// such as the name of your company or organization
	SetRegistryKey(_T("Local AppWizard-Generated Applications"));
	LoadStdProfileSettings(4);  // Load standard INI file options
(including MRU)
	// Register the application's document templates.  Document templates
	//  serve as the connection between documents, frame windows and views
	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(Cmfc1Doc),
		RUNTIME_CLASS(CMainFrame),       // main SDI frame window
```

```
		RUNTIME_CLASS(Cmfc1View));
	if (!pDocTemplate)
		return FALSE;
	AddDocTemplate(pDocTemplate);
	// Parse command line for standard shell commands, DDE, file open
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);
	// Dispatch commands specified on the command line.  Will return FALSE
if
	// app was launched with /RegServer, /Register, /Unregserver or
/Unregister.
	if (!ProcessShellCommand(cmdInfo))
		return FALSE;
	// The one and only window has been initialized, so show and update it
	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();
	// call DragAcceptFiles only if there's a suffix
	//  In an SDI app, this should occur after ProcessShellCommand
	return TRUE;
}



// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	enum { IDD = IDD_ABOUTBOX };

protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

// Implementation
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
END_MESSAGE_MAP()

// App command to run the dialog
void Cmfc1App::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}
```

```
// Cmfc1App message handlers

// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "mfc1.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
      ON_WM_CREATE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
      ID_SEPARATOR,           // status line indicator
      ID_INDICATOR_CAPS,
      ID_INDICATOR_NUM,
      ID_INDICATOR_SCRL,
};

// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
      // TODO: add member initialization code here
}

CMainFrame::~CMainFrame()
{
}

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

      if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
            CBRS_TOP | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY |
            CBRS_SIZE_DYNAMIC) ||
            !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
      {
            TRACE0("Failed to create toolbar\n");
            return -1;      // fail to create
      }

      if (!m_wndStatusBar.Create(this) ||
            !m_wndStatusBar.SetIndicators(indicators,
              sizeof(indicators)/sizeof(UINT)))
```

```
	{
		TRACE0("Failed to create status bar\n");
		return -1;	// fail to create
	}
	// TODO: Delete these three lines if you don't want the toolbar to be
dockable
	m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
	EnableDocking(CBRS_ALIGN_ANY);
	DockControlBar(&m_wndToolBar);

	return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	if( !CFrameWnd::PreCreateWindow(cs) )
		return FALSE;
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return TRUE;
}


// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
	CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
	CFrameWnd::Dump(dc);
}

#endif //_DEBUG


// CMainFrame message handlers

// mfc1Doc.cpp : implementation of the Cmfc1Doc class
//

#include "stdafx.h"
#include "mfc1.h"

#include "mfc1Doc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// Cmfc1Doc

IMPLEMENT_DYNCREATE(Cmfc1Doc, CDocument)

BEGIN_MESSAGE_MAP(Cmfc1Doc, CDocument)
```

```
END_MESSAGE_MAP()


// Cmfc1Doc construction/destruction

Cmfc1Doc::Cmfc1Doc()
{
	// TODO: add one-time construction code here
}

Cmfc1Doc::~Cmfc1Doc()
{
}

BOOL Cmfc1Doc::OnNewDocument()
{
	if (!CDocument::OnNewDocument())
		return FALSE;

	// TODO: add reinitialization code here
	// (SDI documents will reuse this document)

	return TRUE;
}

// Cmfc1Doc serialization

void Cmfc1Doc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
	{
		// TODO: add loading code here
	}
}

// Cmfc1Doc diagnostics

#ifdef _DEBUG
void Cmfc1Doc::AssertValid() const
{
	CDocument::AssertValid();
}

void Cmfc1Doc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
}
#endif //_DEBUG


// Cmfc1Doc commands

// mfc1View.cpp : implementation of the Cmfc1View class
//

#include "stdafx.h"
```

```
#include "mfc1.h"

#include "mfc1Doc.h"
#include "mfc1View.h"
#include ".\mfc1view.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// Cmfc1View

IMPLEMENT_DYNCREATE(Cmfc1View, CView)

BEGIN_MESSAGE_MAP(Cmfc1View, CView)
	// Standard printing commands
	ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
	ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
	ON_WM_LBUTTONDOWN()
END_MESSAGE_MAP()

// Cmfc1View construction/destruction

Cmfc1View::Cmfc1View()
{
	// TODO: add construction code here
}

Cmfc1View::~Cmfc1View()
{
}

BOOL Cmfc1View::PreCreateWindow(CREATESTRUCT& cs)
{
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs

	return CView::PreCreateWindow(cs);
}

// Cmfc1View drawing

void Cmfc1View::OnDraw(CDC* /*pDC*/)
{
	Cmfc1Doc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	if (!pDoc)
		return;

	// TODO: add draw code for native data here
}


// Cmfc1View printing

BOOL Cmfc1View::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
```

```
	return DoPreparePrinting(pInfo);
}

void Cmfc1View::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void Cmfc1View::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}


// Cmfc1View diagnostics

#ifdef _DEBUG
void Cmfc1View::AssertValid() const
{
	CView::AssertValid();
}

void Cmfc1View::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

Cmfc1Doc* Cmfc1View::GetDocument() const // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(Cmfc1Doc)));
	return (Cmfc1Doc*)m_pDocument;
}
#endif //_DEBUG


// Cmfc1View message handlers

void Cmfc1View::OnLButtonDown(UINT nFlags, CPoint point)
{
	// TODO: Add your message handler code here and/or call default
	CClientDC dc(this);
	dc.TextOut(point.x, point.y, "Hello!");
	CView::OnLButtonDown(nFlags, point);
}
```