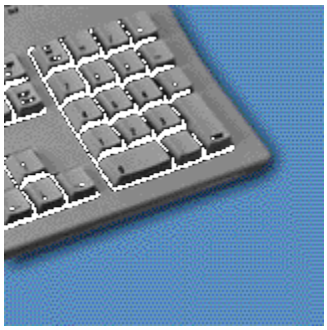


10. RTTI의 원리

<장도비라>
이 장의 주제 RTTI는 6장의 메시지 맵과 더불어 MFC 코드 분석의 핵심이 되는 내용이다. RTTI란 실행시간에 클래스의 타입 정보를 얻어내고, 이 타입의 객체를 생성하는데 사용할 수 있으며, 생성된 객체가 같은 타입인지를 비교하는데도 사용할 수 있다. 이 장에서 우리는 직접 RTTI를 구현할 것이다. 아울러 RTTI를 사용한 MFC 클래스들을 살펴볼 것이다. 이 장이 MFC 코드는 사용하고 있지 않지만, 7장에서 시작한 Single 프로젝트의 소스 분석을 계속하는 장이라는 것을 염두에 두자. 이전 장까지 우리는 차례대로 메시지 맵(7장), DC(8장)와 리소스 에디터(9장)의 원리를 살펴보았다. Single 프로젝트의 RTTI 분석은 11장에서 할 것인데, 여기서는 11장의 소스 분석에 앞서 RTTI의 핵심 부분을 Single 프로젝트와는 별도로 구현해서 개념을 이해하는데 집중할 것이다.
</장도비라>

4장에서 MFC의 RTTI에 사용된 매크로와 CRuntimeClass라는 구조체를 살펴보았다. 이제 그 매크로와 구조체를 이용하여 동적 생성(dynamic creation)과 RTTI(run-time type identification)을 지원하는 윈도우 응용 프로그램을 만들어보자. 이 장에서 살펴볼 내용은 다음과 같다.

- CRuntimeClass를 이용하여 타입에 무관하게 객체 생성하기
- 실행시간에 클래스의 타입 정보를 얻어내기 위한 MFC의 방법
- 베이스 클래스로 사용하는 CObject, CWnd와 CView

CRuntimeClass 구조체

<절도비라>
이 절에서는 RTTI의 기본이 되는 CRuntimeClass 구조체의 원리를 이해

하고 설계해보자. CRuntimeClass는 클래스의 이름과 크기 그리고 객체 생성 함수의 주소를 가지는 작은 구조체이다. MFC는 CRuntimeClass 구조체와 연관된 몇 개의 매크로를 사용하여 객체의 동적 생성에 관한 코드를 자동화하는데, 우리는 그 매크로들을 직접 구현할 것이다.
</절도비라>

MFC의 초기 설계자들은 MFC를 설계하면서 클래스의 동적 생성과 실행시 타입에 관한 정보가 필요하다는 것을 알았다. 하지만 그 당시 이러한 사항들은 C++이 지원하지 않았고, 그들은 당시의 C++ 표준을 이용해서 이러한 것을 직접 구현했다.

MFC의 RTTI는 몇 개의 매크로와 클래스가 지켜야할 규칙을 명시하며, 이러한 조건을 만족하도록 클래스를 설계하면 실행시간에 클래스에 관한 정보와 실행 시간에 이름이 알려지는 클래스의 동적 생성이 가능□하다. 이의 핵심을


<여기서 잠깐>
MFC의 RTTI라고 하면, 동적 생성과 실행시간에 클래스 정보를 얻는 모두를 가리킨다. C++의 RTTI는 동적 생성을 지원하지 않는다.
</여기서 잠깐>

이루는 것이 **CRuntimeClass**라는 구조체이다. 아래는 CRuntimeClass 구조체인데, MFC의 것과 비슷하게 작성한 것이다.

```
struct CRuntimeClass
{
    char        m_lpszClassName[21];
    int         m_nObjectSize;
    CObject* (*pfnCreateObject)();//function pointer

    CObject* CreateObject();
};//struct CRunTimeClass
```

CRuntimeClass 구조체의 첫 번째 멤버 m_lpszClassName[]은 클래스 이름을 의미하는 문자열이다. 두 번째 멤버 m_nObjectSize는 객체의 크기, 세 번째 멤버 pfnCreateObject는 클래스마다 존재해야하는 자기 자신을 생성하는 함수의 시작주소를 가질 것이다. 네 번째 멤버 CreateObject()는 멤버 함수인데 세 번째 멤버에 설정된 함수 포인터를 이용하여 함수를 호출하는 단순한 일을 한다.

CRuntimeClass의 세 번째 멤버와 네 번째 멤버 때문에 동적 생성이 가능한

모든 클래스는 반드시 CObject를 베이스 클래스로 가져야한다. CRuntimeClass구조체의 CreateObject()는 다음과 같이 구현한다.

```
CObject* CRuntimeClass::CreateObject()
{
    return (*pfnCreateObject)();//함수 포인터를 이용하여
                                //간접적으로 함수를 호출한다.
}//CRuntimeClass::CreateObject()
```

CRuntimeClass가 제공된다면 RTTI를 지원하기 위해서 클래스가 지켜야 할 규칙은 다음과 같다.

1) 클래스는 반드시 CRuntimeClass 타입의 static 멤버를 가져야 한다.
2) 1)에 선언된 static 멤버를 초기화하는 루틴을 구현 파일에 가져야 한다.
3) 클래스는 반드시 CObject*를 리턴하는 정적(static) CreateObject() 함수를 선언해야 한다.
4) 3)에 선언된 스태틱 멤버 함수의 몸체를 구현 파일에 가져야 한다.

각각은 다음과 같은 이유 때문에 필요하다.

1) CRuntimeClass는 객체를 생성하기 전, 클래스 레벨에서 접근해야 한다. 그래서 static으로 선언되어야 한다.
2) 초기화 루틴은 클래스 이름, 클래스 크기를 설정하고, CRuntimeClass::pfnCreateObject를 클래스의 정적 멤버 함수 CreateObject로 초기화한다.
3) CreateObject()의 프로토타입은 반드시 static CObject* CreateObject(); 이다. 클래스 레벨에서 접근해야 하므로, 반드시 static 이어야 한다.
4) CreateObject()는 자기 자신을 동적으로 생성하고, 생성된 객체의 시작 주소를 리턴한다. CreateObject()의 리턴 값을 받는 코드는 서브타입의 원리(subtype principle)를 위반하지 않기 위해 반드시 CObject*의 하위 타입이어야 한다. 이것은 동적 생성을 지원하는 클래스가 반드시 CObject를 상속 받아야 하는 이유이다.

1)과 2)의 일을 하는 매크로는 다음과 같다.

1) DECLARE_DYNAMIC(class_name)
2) IMPLEMENT_DYNAMIC(class_name)

DECLARE_DYNCREATE()는 1)과 3)을 한다.
IMPLEMENT_DYNCREATE()는 2)와 4)를 한다.

위와 비슷한 매크로들을 4장에서 살펴보았다. 소스는 아래와 같다.

```
#define RUNTIME_CLASS(class_name) (&class_name::class##class_name)

#define DECLARE_DYNAMIC(class_name) static CRuntimeClass\
 class##class_name;

#define IMPLEMENT_DYNAMIC(class_name) CRuntimeClass \
    class_name::class##class_name = { \
        (#class_name), \
        sizeof(class_name), \
        class_name::CreateObject };

#define DECLARE_DYNCREATE(class_name) \
    DECLARE_DYNAMIC(class_name) \
    static CObject* CreateObject();

#define IMPLEMENT_DYNCREATE(class_name) \
    IMPLEMENT_DYNAMIC(class_name) \
    CObject* class_name::CreateObject() { return new class_name; }
```

DECLARE_DYNAMIC()은 클래스에 CRuntimeClass 타입의 스태틱 멤버를 선언하기 위해 사용한다.

```
DECLARE_DYNAMIC(CTest)
```

처럼 사용한다면,

```
static CRuntimeClass classCTest;
```

처럼 확장되어, classCTest라는 정적 멤버가 선언된다.

RUNTIME_CLASS()는 객체의 CRuntimeClass 타입의 스태틱 멤버를 얻어내기 위해 사용한다.

```
RUNTIME_CLASS(CTest)
```

위의 문장처럼 사용한다면, 아래의 문장처럼 확장되어 CTest 클래스의 정적 멤버 classCTest의 시작 주소를 표현한다.

```
&CTest::classCTest
```

IMPLEMENT_DYNAMIC()은 정적으로 선언된 CRuntimeClass 타입의 멤버를 초기화하기 위해 사용한다.

```
IMPLEMENT_DYNAMIC(CTest)
```

위의 문장처럼 사용한다면, 아래의 문장처럼 초기화된다.

```
CRuntimeClass CTest::classCTest = {
    ("CTest"),
    sizeof(CTest),
    CTest::CreateObject };
```

DECLARE_DYNCREATE()는 DECLARE_DYNAMIC()을 호출하고 추가적인 정적 멤버 함수 CreateObject()를 선언하기 위해 사용한다.

```
DECLARE_DYNCREATE(CTest)
```

위의 문장처럼 사용한다면, 아래의 문장처럼 확장된다.

```
DECLARE_DYNAMIC(CTest)
static CObject* CreateObject();
```

IMPLEMENT_DYNCREATE()는 IMPLEMENT_DYNAMIC()을 호출하고, CreateObject()를 정의하기 위해 사용한다.

```
IMPLEMENT_DYNCREATE(CTest)
```

위의 문장처럼 사용한다면, 아래의 문장처럼 확장된다.

```
IMPLEMENT_DYNAMIC(CTest)
CObject* CTest::CreateObject() {
```

```
    return new CTest;
}
```

CObject 클래스

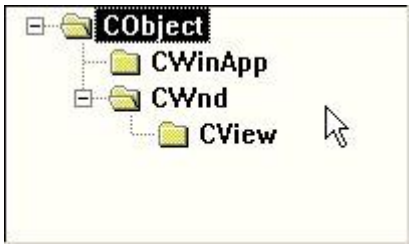
<절도비라>

(이유가 없습니다. 왜 지금 CObject 클래스를 구현해야 하는지 서술해주세요..)

객체의 동적 생성을 위해서는 CRuntimeClass 객체의 주소를 얻어야 하고, 이를 통해 생성된 객체는 그 타입이 알려지지 않았으므로, 동적 생성 객체들의 공통 조상인 CObject*로 리턴된다. 이 절에서는 CObject 클래스를 구현한다.

</절도비라>

MFC에서 CObject는 거의 대부분 클래스의 공통 조상(root class)이다. 우리는 MFC의 CObject와 비슷하게 CObject를 구현한다. 이 장에서 구현할 클래스 계층도는 아래 [그림 10.1]과 같다.



[그림 10.1] CObject의 하위클래스들: CObject는 모든 클래스의 베이스 클래스이다.

CObject는 몇 개의 가상함수를 포함하고, 디버깅을 위해 operator new()을 오버로딩하는 간단한 클래스이다. 설계한 CObject는 아래 [예제 10.1]과 같다.

[예제 10.1] class CObject

```
class CObject
{
public:
```

```
    virtual CRuntimeClass* GetRuntimeClass() const;
    //virtual void Serialize(CArchive& ar);
    virtual ~CObject(){}

protected:
    CObject(){}
};//class CObject
```

CObject는 베이스 클래스로 사용되므로, 파괴자는 가상으로 선언되었고, 동적 생성을 위해 CRuntimeClass를 얻어내는 멤버 함수를 가상으로 선언하였다.

이 장의 목적은 RTTI의 원리를 이해하는 것이다. 그러기 위해서 우리는 MFC의 다큐먼트/뷰 구조와 유사한 클래스들을 차례대로 설계해 볼 것이고, 그 클래스들이 RTTI를 지원하도록 할 것이다.

CWnd

<절도비라>

이 절에서는 객체의 동적 생성을 지원하는 첫 클래스로, 윈도우의 베이스 클래스로 사용되는 CWnd 클래스를 구현한다. CWnd는 7장에서 작성한 Single 프로젝트의 CWnd에 대응하는 클래스이다. 우리는 후에 CWnd를 상속받아 다큐먼트/뷰 구조의 뷰 파트를 만들 것이다. 우리는 또한 CWnd가 GetSafeHwnd()를 가지는 이유에 대해서도 살펴볼 것이다..

</절도비라>

MFC의 CWnd 클래스는 모든 윈도우 클래스의 베이스 클래스로 사용된다. 그러므로 윈도우 핸들을 필요로 하는 모든 윈도우 API함수들을 랩(wrap)하고 있고, 구조상 필요한 가상 함수들을 포함한다. 구현할 CWnd의 헤더는 다음 [예제 10.2]와 같이 구성된다.

[예제 10.2] class CWnd

```
class CWnd : public CObject
{
public:
```

```
    DECLARE_DYNCREATE(CWnd) // (1)

    HWND m_hwnd; // (2)

public:
    HWND GetSafeHwnd(); // (3)
    BOOL ShowWindow(int nCmdShow); // (4)
    BOOL UpdateWindow();

    //{{AFX_VIRTUAL
    virtual void PreCreateWindow(CREATESTRUCT& cs); // (5)
    //}}AFX_VIRTUAL
};//class CView
```

CWnd는 (1) 동적 생성이 가능하고, (2) 윈도우 핸들을 멤버로 가지며, (3) 윈도우 핸들을 얻어오는 함수를 제공한다. 또한, (4) API함수 ShowWindows()를 감싼 멤버함수를 제공하고, (5) 윈도우 유지에 필요한 가상 함수를 제공한다.

PreCreateWindow()는 대표적인 가상 함수인데 윈도우를 생성하기 바로 전에 호출된다. 그러므로 MFC에서 윈도우 스타일을 변경하기 위해서는 PreCreateWindow()를 오버라이드한다. 예에서는 호출만 적절하게 수행하고, PreCreateWindow()가 동작하도록 구현하지는 않았다. 독자들이 PreCreateWindow()의 변경이 영향을 미치도록 꼭 소스를 수정해 보기 바란다.

윈도우 핸들을 얻어오는 함수가 GetHwnd()가 아니고, 왜 GetSafeHwnd()인가? 왜 안전하지 않은 경우가 존재하는가? 아래를 읽지 말고 독자들 스스로 답을 해 보기 바란다.

이유는 GetSafeHwnd()의 MFC 소스를 보면 명확해 진다. 이 함수를 호출했을 때, 객체가 존재하지 않는 경우, 즉 this가 NULL인 경우가 존재한다. 왜 그런가? 그것은 객체의 동적 생성 때문이다. CWnd를 상속받아 구현된 윈도우 클래스의 객체가 만들어지는 시점은 뷰를 포함한 대부분의 경우, CWinApp클래스의 InitInstance()에서이다. MFC는 뷰의 관리를 위해 뷰의 윈도우 핸들을 얻는 함수를 여러 곳에서 호출할 것이고, 객체가 생성되지 않은 어느 시점에서 이 함수를 호출했을 때, this가 NULL이라면 윈도우 핸들은 유효하지 않다. 그래서 윈도우 핸들이 안전한지 어떤지를 검사할 필요가 있다.

그래서 GetSafeHwnd()의 소스는 아래와 같다.

```
HWND CWnd::GetSafeHwnd()
{
    return this == NULL ? NULL : m_hwnd;
}//CWnd::GetSafeHwnd()
```

품을 수 있는 한 가지 의문은 this가 NULL인데 멤버 함수의 호출이 가능한가? 하는 것이다. 가능하다. 왜냐하면 this는 멤버 함수 호출에서 파라미터로 전달되는 첫 번째 파라미터이기 때문이다. 그래서 this가 NULL인지 검사를 하지 않고, m_hwnd를 접근하면 [0]번지에서 m_hwnd의 상대 주소를 접근할 것이고, 그것은 GPF(general protection fault)를 발생한다.

CWinApp

<절도비라>
이 절에서는 7장에서 작성한 Single 프로젝트의 CSingleApp에 대응하는 CWinApp를 설계한다. CWinApp는 응용 프로그램의 초기화, 다큐먼트와 뷰의 결합등을 담당한다. CWinApp는 다큐먼트와 뷰의 템플릿 결합을 담당하는 멤버를 가지며, 메인 프레임 윈도우에 대한 포인터를 가진다. 이 절에서는 추가적으로 MFC의 가상 함수 OnIdle()의 원리를 이해하고, OnOdle()을 지원하도록 메시지 루프를 변경할 것이다.
</절도비라>

CWinApp는 지금껏 구현했던 기존의 코드에 메인 윈도우 핸들과 메인 윈도우를 생성하는 AddDocTemplate(), 그리고 아이들 시간(idle time)을 처리하는 가상 함수 OnOdle()을 가진다. MFC 응용 프로그램은 메인 프레임 윈도우와 클라이언트 영역을 나타내는 뷰 윈도우를 별도로 구현했지만, 우리는 프레임 윈도우와 뷰를 보두 CView로 구현하기로 하자. 클래스 선언은 다음 [예제 10.3]과 같다.

[예제 10.3] class CWinApp

```
class CWinApp : public CObject
```

```
{
protected:
    static char     szAppName[];
    HINSTANCE       m_hInstance;

public:
    CWnd*           m_pMainWnd; // (1)

public:
    virtual ~CWinApp();
    void AddDocTemplate(CRuntimeClass* pRuntimeClass); // (2)
    void InitInstance(HINSTANCE hInstance, PSTR szCmdLine,
                      int iCmdShow);
    void Run();
    WPARAM ExitInstance();

    // virtual function
    virtual int OnIdle(); // Note: OnIdle() is virtual // (3)
};//class CWinApp
```

m_pMainWnd는 응용 프로그램의 메인 윈도우 객체를 가리킨다(1). 이 포인터는 AddDocTemplate()에서 초기화된다(2). AddDocTemplate()가 받는 파라미터에 주목하자. CRuntimeClass 구조체의 포인터를 받으므로, 전달되는 어떠한 객체라도 동적 생성이 가능하다! AddDocTemplate()의 소스는 [예제 10.4]와 같다.

[예제 10.4] AddDocTemplate()

```
void CWinApp::AddDocTemplate(CRuntimeClass* pRuntimeClass)
{
    m_pMainWnd = (CWnd*)pRuntimeClass->CreateObject(); // (1)
    if ( m_pMainWnd == NULL )
        return;

    CREATESTRUCT    cs;

    m_pMainWnd->PreCreateWindow( cs ); // (2)
    m_pMainWnd->m_hwnd = ::CreateWindow(
        szAppName,              //window class name
        "RTTI",                 //window caption
        WS_OVERLAPPEDWINDOW,    //window style
```

```
        CW_USEDEFAULT,          //initial x position
        CW_USEDEFAULT,          //initial y position
        CW_USEDEFAULT,          //initial x size
        CW_USEDEFAULT,          //initial y size
        NULL,                   //parent window handle
        NULL,                   //window menu handle
        m_hInstance,            //program instance handle
        NULL );                 //creation parameters
}//CWinApp::AddDocTemplate()
```

먼저 전달된 CRuntimeClass를 만들고, 포인터를 m_pMainWnd에 대입한다(1). 그리고, 메인 윈도우의 가상함수 PreCreateWindow()를 호출하고, API함수 CreateWindow()를 호출한다(2). PreCreateWindow()에서의 변경된 사항은 CreateWindow()에 반영된다.

가능한 한 가지 질문은 함수의 이름이 왜 AddDocTemplate()인가 하는 것이다. 이것은 MFC의 소스를 보면 명확해 진다. 싱글 다큐먼트(single document) 프로젝트의 경우 이 함수는 아래와 같다.

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

하지만, 멀티플 다큐먼트(multiple document) 프로젝트의 경우 이 함수는 아래와 같다.

```
CMultiDocTemplate* pDocTemplate;
pDocTemplate = new CMultiDocTemplate(
    IDR_MULTITYPE,
    RUNTIME_CLASS(CMultiDoc),
    RUNTIME_CLASS(CChildFrame), // custom MDI child frame
    RUNTIME_CLASS(CMultiView));
AddDocTemplate(pDocTemplate);

// create main MDI Frame window
```

```
    CMainFrame* pMainFrame = new CMainFrame;
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))
        return FALSE;
    m_pMainWnd = pMainFrame;
```

AddDocTemplate()은 동적으로 다큐먼트, 프레임과 뷰를 만들고 관련된 멤버를 초기화한다. 이러한 상관 관계는 싱글 다큐먼트인 경우와 멀티플 다큐먼트인 경우가 다른데, MFC에는 모두 두 개의 다큐먼트 구조(document template)가 존재한다. AddDocTemplate()이 실제로 받는 파라미터는 모든 다큐먼트 템플릿의 베이스 클래스인 CDocTemplate의 시작 주소이다. 그러므로 멤버 함수의 이름이 AddDocTemplate()이다.

InitInstance()는 윈도우 클래스를 등록하고(1), AddDocTemplate()를 호출해서 동적으로 메인 윈도우 객체를 생성한다(2). 그리고 생성된 메인 윈도우 객체의 ShowWindow()와 UpdateWindow()를 호출한다(3). 소스는 아래 [예제 10.5]와 같다.

[예제 10.5] InitInstance()

```
void CWinApp::InitInstance(HINSTANCE    hInstance,
                           PSTR         szCmdLine,
                           int          iCmdShow)
{
    WNDCLASSEX  wndclass;

    wndclass.cbSize         = sizeof(wndclass);
    wndclass.style          = CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc    = WndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szAppName;
    wndclass.hIconSm        = LoadIcon(NULL,IDI_APPLICATION);

    ::RegisterClassEx(&wndclass); // (1)
```

```
    m_hInstance = hInstance;

    AddDocTemplate( RUNTIME_CLASS(CView) ); // (2)

    m_pMainWnd->ShowWindow( iCmdShow ); // (3)
    m_pMainWnd->UpdateWindow();
}//CWinApp::InitInstance
```

이제 아이들 타임을 지원하는 메시지 루프를 완성해 보자. 작성할 메시지 루프는 메시지 큐에 메시지가 없는 경우, OnIdle()을 호출하고자 하는 것이다. 소스는 아래 [예제 10.6]과 같다.

[예제 10.6] Run()

```
void CWinApp::Run()
{
    MSG msg;
    int bFlag = TRUE;

    while ( TRUE )
    {
        if ( PeekMessage(&msg, NULL, 0, 0, PM_REMOVE) )
        {
            if ( msg.message == WM_QUIT )
                break;

            TranslateMessage( &msg );
            DispatchMessage( &msg );
            bFlag = TRUE;
        }//if

        if ( bFlag )
            bFlag = OnIdle();
    }//while
}//CWinApp::Run()
```

소스를 보면 OnIdle()이 TRUE를 리턴 했을 경우에만 다음 아이들 시간에 OnIdle()이 호출되는 것을 알 수 있다.

여기서 독자들에게 문제를 한 가지 내겠다. 위 메시지 루프의 문제점은 무

엇인가? 문제점을 지적하고 해결하라.

OnIdle()이 제대로 동작하는지 확인하기 위해서 OnIdle()을 [예제 10.7]처럼 구현해 보자.

[예제 10.7] OnIdle()의 구현

```
int CWinApp::OnIdle()
{
    HWND   hwnd = m_pMainWnd->GetSafeHwnd(); // (1)
    HDC    hdc;
    RECT   rect, rect2;
    HBRUSH hBrush[3];

    hBrush[0] = CreateSolidBrush( RGB(255,0,0) );
    hBrush[1] = CreateSolidBrush( RGB(0,255,0) );
    hBrush[2] = CreateSolidBrush( RGB(0,0,255) );

    GetClientRect( hwnd, &rect );

    hdc = GetDC( hwnd );

    rect2.left   = rand() % rect.right;
    rect2.top    = rand() % rect.bottom;
    rect2.right  = rand() % rect.right;
    rect2.bottom = rand() % rect.bottom;

    FillRect( hdc, &rect2, hBrush[rand() % 3] ); // (2)

    ReleaseDC( hwnd, hdc );

    DeleteObject( hBrush[0] );
    DeleteObject( hBrush[1] );
    DeleteObject( hBrush[2] );

    return FALSE; // (3)
    //return TRUE; //The difference!
}//CWinApp::OnIdle()
```

OnIdle()은 메인 윈도우를 얻어 내어(1), 클라이언트 영역에 랜덤한 사각형을 그린다(2). OnIdle()이 TRUE를 리턴하면 OnIdle()은 계속해서 호출되지만, FALSE를 리턴하면 첫 번째 아이들 시간에만 호출된다.

여기서 독자들에게 문제를 한 가지 더 낸다. MFC에서 OnIdle()을 위와 같이 구현하면 거의 똑같이 동작한다. 하지만 한 가지 문제점이 발견될 것이다. 그 문제점을 해결하라.

CView

<절도비라>
다큐먼트/뷰 구조의 뷰를 담당하는 CView 클래스를 설계한다. CView는 7장에서 작성한 Single 프로젝트의 CSingleView에 대응한다. 필자는 다큐먼트에 해당하는 부분은 구현하지 않을 것인데 그것은 CView의 구현과 많은 부분이 중복되기 때문이다. CView와 중복되지 않는 다큐먼트의 부분은 13장에서 설명할 것이다.
</절도비라>

CView 클래스는 이전 버전과 거의 같다. 아래는 CView 클래스의 헤더이다.

```
class CView : public CWnd
{
public:
    DECLARE_DYNCREATE(CView) // (1)

public:
    //{{AFX_VIRTUAL
    // you may override some virtual functions
    // you know, there are two types of message map
    //
    //virtual void PreCreateWindow(CREATESTRUCT& cs); // (2)
    //}}AFX_VIRTUAL

    //{{AFX_MESSAGE_MAP // (3)
    void OnDraw();
    void OnDestroy();
    void OnLButtonDown();
    //}}AFX_MESSAGE_MAP

    DECLARE_MESSAGE_MAP()
```

```
};//class CView
```

뷰 클래스는 CWnd를 받아 구현한다. 이전 버전과의 가장 큰 차이점은 동적 생성을 지원한다는 것이다(1), 윈도우와 관련된 가상 함수를 오버라이드 할 수 있으며(2), 메시지 맵을 포함한다(3).

자 어떤가? 모든 것이 이해되었는가? 다음 절에서 프로젝트의 소스를 모두 리스트한다.

프로젝트의 전체 소스

<절도비라>
(ㅠ.ㅠ 냉정하게 이런 도비라 텍스트라면 사실 의미가 없습니다. 지금까지 각 부분을 어떻게 살펴봤는지, 그럼 이 전체 소스를 독자가 보면서 주안점을 두어야 할 부분은 무엇인지를 언급해주셔야 합니다.)
이 절에서는 RTTI 프로젝트의 전체 소스를 살펴본다. 소스에서 주의 깊게 살펴볼 내용은 CObject를 상속 받는 클래스들의 동적 생성에 관한 것이다. 이 클래스는 CWnd와 CView이다. 11장에서 우리는 Single 프로젝트의 동적 생성 부분에 관한 코드를 분석할 것인데, 이와 연관된 클래스는 CSingleDoc, CSingleView와 CMainFrame이다. 동적 생성의 원리는 세 클래스 모두 동일하므로, 11장에서는 CMainFrame의 RTTI만 분석해 볼 것인데 그 원리가 이 절에서 제시한 소스의 CView에 사용된 것과 동일하므로 이 절의 소스를 주의 깊게 읽어두기 바란다. 아울러 다큐먼트 템플릿을 구성하는 부분과 가상 함수 OnOdle()의 원리도 기억해 두자.
</절도비라>

프로젝트의 전체 소스를 아래에 리스트하였다. 이해가 되지 않는다면 반복해서 학습해서 꼭 자기 것으로 만들기 바란다. 소스는 MFC의 핵심 부분의 원리를 대부분 포함하고 있다. 이 프로젝트에 포함되지 않은 DDX와 직렬화는 다음 장들에서 별도로 원리를 알아보고 구현해 볼 것이다.

[예제 10.8] stdafx.h

```
#include <windows.h>
```

```
#ifndef _stdafx_defined_
#define _stdafx_defined_

//{{MessageMap macros-----------------------------------------
#define DECLARE_MESSAGE_MAP()          static MessageMap messageMap[];
#define BEGIN_MESSAGE_MAP(class_name)  MessageMap\
    class_name::messageMap[]={
#define END_MESSAGE_MAP()              {0,NULL}};

//{{RTTI macros-----------------------------------------------
#define RUNTIME_CLASS(class_name) (&class_name::\
    class##class_name)

#define DECLARE_DYNAMIC(class_name) static CRuntimeClass\
    class##class_name;

#define IMPLEMENT_DYNAMIC(class_name) CRuntimeClass \
    class_name::class##class_name = { \
        (#class_name), \
        sizeof(class_name), \
        class_name::CreateObject };

#define DECLARE_DYNCREATE(class_name) \
    DECLARE_DYNAMIC(class_name) \
    static CObject* CreateObject();

#define IMPLEMENT_DYNCREATE(class_name) \
    IMPLEMENT_DYNAMIC(class_name) \
    CObject* class_name::CreateObject() { return new class_name; }

//Forward declaration-----------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd,UINT iMsg,WPARAM wParam,
                         LPARAM lParam);
class CObject;

//{{struct CRuntimeClass--------------------------------------
struct CRuntimeClass
{
    char        m_lpszClassName[21];
    int         m_nObjectSize;
    CObject* (*pfnCreateObject)();//function pointer

    CObject* CreateObject();
```

```
};//struct CRunTimeClass

//Forward declaration----------------------------------------
class CView;

//{{struct MessageMap----------------------------------------
typedef void (CView::*CViewFunPointer)();
typedef struct tagMessageMap
{
    UINT            iMsg;
    CViewFunPointer fp;
} MessageMap;

#endif // #ifndef _stdafx_defined_
```

[예제 10.9] stdafx.cpp

```
#include "stdafx.h"

CObject* CRuntimeClass::CreateObject()
{
    return (*pfnCreateObject)();//함수 포인터를 이용하여
                                //간접적으로 함수를 호출한다.
}//CRuntimeClass::CreateObject()
```

[예제 10.10] CObject.h

```
#include "stdafx.h"

#ifndef _CObject_
#define _CObject_

class CObject
{
public:
    virtual CRuntimeClass* GetRuntimeClass() const;
    //virtual void Serialize(CArchive& ar);
    virtual ~CObject(){}
```

```
protected:
    CObject(){}
};//class CObject

#endif // #ifndef _CObject_
```

[예제 10.11] CObject.cpp

```
#include "CObject.h"

/*virtual*/ CRuntimeClass* CObject::GetRuntimeClass() const
{
    return NULL; // RUNTIME_CLASS(CObject);
}//CObject::GetRuntimeClass()
```

[예제 10.12] CWnd.h

```
#include "stdafx.h"
#include "CObject.h"

#ifndef _CWnd_
#define _CWnd_

class CWnd : public CObject
{
public:
    DECLARE_DYNCREATE(CWnd)

    HWND m_hwnd;

public:
    HWND GetSafeHwnd();
    BOOL ShowWindow(int nCmdShow);
    BOOL UpdateWindow();

    //{{AFX_VIRTUAL
    virtual void PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL
};//class CWnd
```

```
#endif // #ifndef _CWnd_
```

[예제 10.13] CWnd.cpp

```
#include "CWnd.h"

IMPLEMENT_DYNCREATE(CWnd)

HWND CWnd::GetSafeHwnd()
{
    return this == NULL ? NULL : m_hwnd;
}//CWnd::GetSafeHwnd()

//HINSTANCE CWnd::GetInstanceHandle()
//{
//    return (HINSTANCE)GetWindowLong( GetSafeHwnd(), GWL_HINSTANCE );
//}

BOOL CWnd::ShowWindow(int nCmdShow)
{
    return ::ShowWindow( GetSafeHwnd(), nCmdShow );
}//CWnd::ShowWindow(int nCmdShow)

BOOL CWnd::UpdateWindow()
{
    return ::UpdateWindow( GetSafeHwnd() );
}//CWnd::UpdateWindow()

/*virtual*/ void CWnd::PreCreateWindow(CREATESTRUCT& cs)
{
}//CWnd::PreCreateWindow(CREATESTRUCT& cs)
```

[예제 10.14] CWinApp.h

```
#include "stdafx.h"
#include "CObject.h"
#include "CView.h"
```

```
#ifndef _CWinApp_
#define _CWinApp_

//Class CWinApp-----------------------------------------
class CWinApp : public CObject
{
protected:
    static char     szAppName[];
    HINSTANCE       m_hInstance;

public:
    CWnd*           m_pMainWnd;

public:
    virtual ~CWinApp();
    void AddDocTemplate(CRuntimeClass* pRuntimeClass);
    void InitInstance(HINSTANCE hInstance, PSTR szCmdLine,
                      int iCmdShow);
    void Run();
    WPARAM ExitInstance();

    // virtual function
    virtual int OnIdle(); // Note: OnIdle() is virtual
};//class CWinApp

#endif // #ifndef _CWinApp_
```

[예제 10.15] CWinApp.cpp

```
#include "CWinApp.h"

// destructor
CWinApp::~CWinApp()
{
    if ( m_pMainWnd != NULL )
    {
        delete m_pMainWnd;
        m_pMainWnd = NULL;
    }//if
}//CWinApp::~CWinApp()
```

```
void CWinApp::AddDocTemplate(CRuntimeClass* pRuntimeClass)
{
    m_pMainWnd = (CWnd*)pRuntimeClass->CreateObject();
    if ( m_pMainWnd == NULL )
        return;

    CREATESTRUCT    cs;

    m_pMainWnd->PreCreateWindow( cs );
    m_pMainWnd->m_hwnd = ::CreateWindow(
        szAppName,              //window class name
        "RTTI",                 //window caption
        WS_OVERLAPPEDWINDOW,    //window style
        CW_USEDEFAULT,          //initial x position
        CW_USEDEFAULT,          //initial y position
        CW_USEDEFAULT,          //initial x size
        CW_USEDEFAULT,          //initial y size
        NULL,                   //parent window handle
        NULL,                   //window menu handle
        m_hInstance,            //program instance handle
        NULL );                 //creation parameters
}//CWinApp::AddDocTemplate()

void CWinApp::InitInstance(HINSTANCE    hInstance,
                           PSTR         szCmdLine,
                           int          iCmdShow)
{
    WNDCLASSEX  wndclass;

    wndclass.cbSize         = sizeof(wndclass);
    wndclass.style          = CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc    = WndProc;
    wndclass.cbClsExtra     = 0;
    wndclass.cbWndExtra     = 0;
    wndclass.hInstance      = hInstance;
    wndclass.hIcon          = LoadIcon(NULL,IDI_APPLICATION);
    wndclass.hCursor        = LoadCursor(NULL,IDC_ARROW);
    wndclass.hbrBackground  = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wndclass.lpszMenuName   = NULL;
    wndclass.lpszClassName  = szAppName;
    wndclass.hIconSm        = LoadIcon(NULL,IDI_APPLICATION);

    ::RegisterClassEx(&wndclass);
```

```
    m_hInstance = hInstance;

    AddDocTemplate( RUNTIME_CLASS(CView) );

    m_pMainWnd->ShowWindow( iCmdShow );
    m_pMainWnd->UpdateWindow();
}//CWinApp::InitInstance

void CWinApp::Run()
{
    MSG msg;
    int bFlag = TRUE;

    while ( TRUE )
    {
        if ( PeekMessage(&msg, NULL, 0, 0, PM_REMOVE) )
        {
            if ( msg.message == WM_QUIT )
                break;

            TranslateMessage( &msg );
            DispatchMessage( &msg );
            bFlag = TRUE;
        }//if

        if ( bFlag )
            bFlag = OnIdle();
    }//while
}//CWinApp::Run()

WPARAM CWinApp::ExitInstance()
{
    return 0L;//msg.wParam;
}//CWinApp::ExitInstance

int CWinApp::OnIdle()
{
    HWND   hwnd = m_pMainWnd->GetSafeHwnd();
    HDC    hdc;
    RECT   rect, rect2;
    HBRUSH hBrush[3];
```

```
    hBrush[0] = CreateSolidBrush( RGB(255,0,0) );
    hBrush[1] = CreateSolidBrush( RGB(0,255,0) );
    hBrush[2] = CreateSolidBrush( RGB(0,0,255) );

    GetClientRect( hwnd, &rect );

    hdc = GetDC( hwnd );

    rect2.left   = rand() % rect.right;
    rect2.top    = rand() % rect.bottom;
    rect2.right  = rand() % rect.right;
    rect2.bottom = rand() % rect.bottom;

    FillRect( hdc, &rect2, hBrush[rand() % 3] );

    ReleaseDC( hwnd, hdc );

    DeleteObject( hBrush[0] );
    DeleteObject( hBrush[1] );
    DeleteObject( hBrush[2] );

    return FALSE;
    //return TRUE; //The difference!
}//CWinApp::OnIdle()

char CWinApp::szAppName[]="HelloWin";

//Global object-----------------------------------------
extern CWinApp app;

//Window procedure--------------------------------------
LRESULT CALLBACK WndProc(HWND hwnd, UINT iMsg, WPARAM wParam,
                         LPARAM lParam)
{
    static CViewFunPointer  fpCViewGlobal = NULL;
        //pointer to a member function
    int                     i = 0;
    CView*                  p;

    while ( CView::messageMap[i].iMsg != 0 )
    {
        if ( iMsg == CView::messageMap[i].iMsg )
        {
```

```
            fpCViewGlobal = CView::messageMap[i].fp;
            p = static_cast<CView*>(app.m_pMainWnd);
            (p->*fpCViewGlobal)();

            return 0;
        }//if
        ++i;
    }//while

    return DefWindowProc(hwnd,iMsg,wParam,lParam);
}//WndProc()

int WINAPI WinMain(HINSTANCE    hInstance,
                   HINSTANCE    hPrevInstance,
                   PSTR         szCmdLine,
                   int          iCmdShow)
{
    app.InitInstance( hInstance, szCmdLine, iCmdShow );
    app.Run();

    return app.ExitInstance();
}//WinMain()
```

[예제 10.16] CView.h

```
#include "stdafx.h"
#include "CWnd.h"

#ifndef _CView_
#define _CView_

class CView : public CWnd
{
public:
    DECLARE_DYNCREATE(CView)

public:
    //{{AFX_VIRTUAL
    // you may override some virtual functions
    // you know, there are two types of message map
    //
```

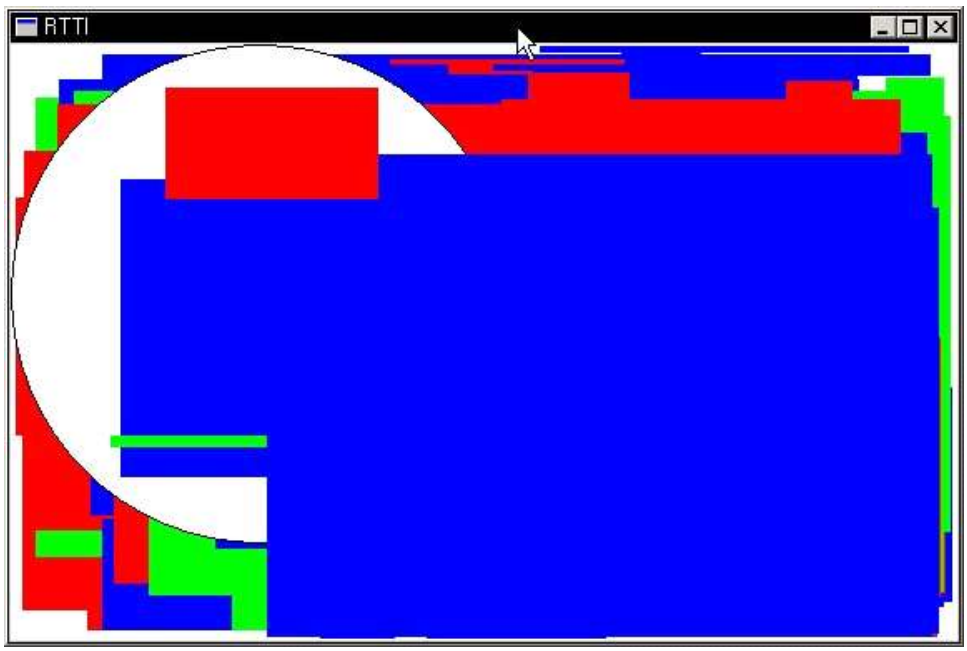
```
    //virtual void PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

    //{{AFX_MESSAGE_MAP
    void OnDraw();
    void OnDestroy();
    void OnLButtonDown();
    //}}AFX_MESSAGE_MAP

    DECLARE_MESSAGE_MAP()
};//class CView

#endif // #ifndef _CView_
```

[예제 10.17] CView.cpp

```
#include "CWinApp.h"
#include "CView.h"

IMPLEMENT_DYNCREATE(CView)

BEGIN_MESSAGE_MAP(CView)
    { WM_PAINT,       CView::OnDraw },
    { WM_DESTROY,     CView::OnDestroy },
    { WM_LBUTTONDOWN, CView::OnLButtonDown },
END_MESSAGE_MAP()

CWinApp app;

//CView Event handler-----------------------------------------
void CView::OnDraw()
{
    HDC             hdc;
    PAINTSTRUCT     ps;
    RECT            rect;

    hdc = BeginPaint( GetSafeHwnd(), &ps );
    GetClientRect( GetSafeHwnd(), &rect );
    DrawText( hdc, "Hello, Windows!", -1, &rect,
        DT_SINGLELINE|DT_CENTER|DT_VCENTER );
    EndPaint( GetSafeHwnd(), &ps );
}//CView::OnDraw()
```

```
void CView::OnDestroy()
{
    PostQuitMessage(0);
}//CView::OnDestroy()

void CView::OnLButtonDown()
{
    HDC hdc;

    hdc = GetDC( GetSafeHwnd() );
    Ellipse( hdc, 0, 0, 300, 300 );
    ReleaseDC( GetSafeHwnd(), hdc );
}//CView::OnLButtonDown()
```

실행 결과는 아래 그림과 같다.



[그림 10.2] 예제의 실행화면: 클라이언트 영역에서 마우스를 움직여 보자. 마우스 메시지와 메시지 사이에서 랜덤한 사각형이 그려지는 것을 확인할 수 있다.

클라이언트 영역에서 마우스 커서를 움직이면 잠시 동안 랜덤한 사각형이

그려지는 것을 확인할 수 있다.

(프로그램 설명은 마쳤습니다. 그럼 이 장의 내용을 모두 소화한 독자가 해야 할 일은 무엇일까요? 그러한 방향을 짚어주는 것도 좋은 서비스입니다.)

다음 장에서 우리는 7장 Single 프로젝트의 RTTI부분을 분석할 것이고, RTTI가 이 장에서 구현해 본 것과 똑같은 방법으로 구현되어 있는 것을 확인할 수 있을 것이다. Single 프로젝트의 모든 분석은 13장에서 마칠 것이다.

요약

객체 동적 생성의 핵심은 CRuntimeClass 구조체를 스태틱으로 포함하는 것이다. MFC는 CRuntimeClass와 연관된 몇 개의 매크로를 제공함으로써 클래스에 RTTI 기능을 추가한다.

OnIdle()은 처리할 메시지가 없을 때 호출되도록 설계된 가상함수이다.

- **CRuntimeClass**는 객체의 동적 생성을 위한 핵심 구조체로 객체 생성을 위한 함수 포인터와 CreateObject()를 멤버 함수로 가진다.
- **RUNTIME_CLASS()** 매크로는 클래스의 정적 CRuntimeClass 멤버의 시작 주소를 표현한다.
- **CObject**는 대부분 MFC 클래스들의 조상으로 사용된다.
- **CWnd**는 모든 윈도우 클래스의 조상으로 사용된다.
- **CView**는 윈도우 클라이언트 영역에 대응하는 윈도우 클래스로 데이터를 표현하는 역할을 한다.

[문서의 끝]