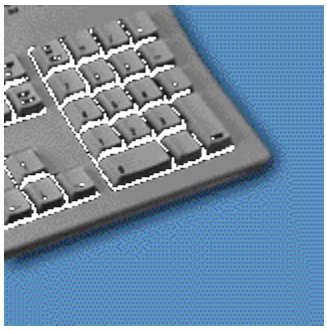


9. 리소스 편집

<장도비라>
우리는 이 장에서 비주얼 C++의 리소스 편집기의 원리를 이해하고, 텍스트 편집기를 이용하여 직접 리소스 스크립트를 편집하는 방법을 살펴볼 것이다. 사실 리소스 편집기와 MFC는 독립적이다. 즉 MFC 프로젝트가 아니더라도 리소스 편집기는 똑같이 동작한다. 하지만, MFC 프로젝트는 반드시 리소스를 사용하므로, 이러한 리소스를 관리하는 비주얼 C++의 방법을 살펴보는 것은 많은 도움이 될 것이다. 우리는 아이콘과 메뉴 리소스를 프로그램에 추가하여 구현할 것이다. 아울러 MFC 코드 분석의 리듬을 잃지 않기 위해, 7장에서 시작한 Single 프로젝트에서 사용된 MFC의 가상함수들도 살펴볼 것이다.
</장도비라>

이 장에서 비주얼 C++의 리소스 편집기(resource editor)의 원리를 알아본다. 원리를 이해하는 것이 목적이므로, 비주얼 C++의 리소스 편집기를 사용하지 않을 것이고, 리소스 스크립트(resource script)를 텍스트 편집기를 사용하여 직접 입력할 것이다. 이 장에서 살펴볼 내용은 아래와 같다.

- 리소스를 기술하는 언어인 리소스 스크립트
- 비주얼 리소스 에디터의 원리
- 텍스트 에디터로 리소스 스크립트를 편집하기
- 비주얼 스튜디오가 리소스 ID를 관리하는 방법
- 이벤트에 대응하는 MFC의 가상함수들

리소스 스크립트

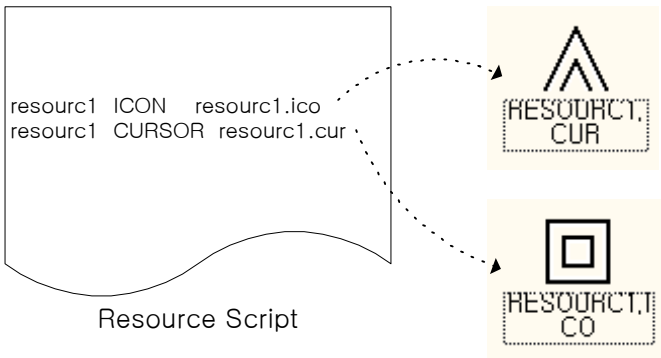
<절도비라>
이 절에서 우리는 비주얼 C++의 리소스 에디터를 사용하여 아이콘을 만들고, 이를 프로그램에서 사용하는 간단한 응용 프로그램을 작성한다. 이를

통해 윈도우즈 리소스를 기술하는 언어인 리소스 스크립트에 대해서 살펴본다.
</절도비라>

윈도우 응용 프로그램을 만드는 과정은 기존의 컴파일(compile), 링크(link)에 추가적인 **리소스 컴파일(resource compile)**이라는 과정을 거친다. 리소스는 생성된 응용 프로그램의 코드 - 코드 블록과 데이터 블록을 포함한 - 를 제외한 나머지 부분이다. 그것은 메뉴(menu), 비트맵(bitmap), 아이콘(icon), 커서(cursor) 등 대부분 UI(user interface)를 위한 것들이다. 윈도우즈는 이러한 리소스를 기술하는 특별한 언어를 제공하는데 이것을 **리소스 스크립트(resource script)**□라 한다.

<여기서 잠깐>

스크립트는 프로그래밍 언어의 일종이지만, 특별한 응용 프로그램에서만 사용하거나, 실행 가능한 파일이 만들어지지 않는 다는 면에서 일반적인 프로그래밍 언어와는 다르다. 널리 알려진 자바 스크립트(Java script)를 보면, HTML에 포함되어, IE(internet explorer)의 입력으로 사용되지만, 단독으로 실행할 수는 없다.
</여기서 잠깐>



[그림 9.1] 리소스 스크립트: 리소스 스크립트에는 아이콘, 커서 등 외부 파일을 지정할 수 있다. 또한, 스크립트 자체로도 대화상자나 메뉴 등의 리소스를 기술한다.

리소스 스크립트로 대화상자(dialog box)나 메뉴 등을 기술할 수 있다. 또한, 외부의 파일을 명시할 수 있는데, 그러한 것들에는 아이콘, 커서, 비트맵 등이 있다. 윈도우즈용 컴파일러는 리소스 스크립트와 스크립트에서 지정한 파일을 입력으로 받아 들여, 이진 리소스(binary resource)를 생성하는 특별한 리소스 컴파일러를 제공한다.

아래의 예는 첫 번째 예에서 사용할 간단한 리소스 스크립트인데, 외부에

존재하는 resourc1.ico 아이콘 파일과, resourc1.cur라는 커서 파일을 명시한 것이다.

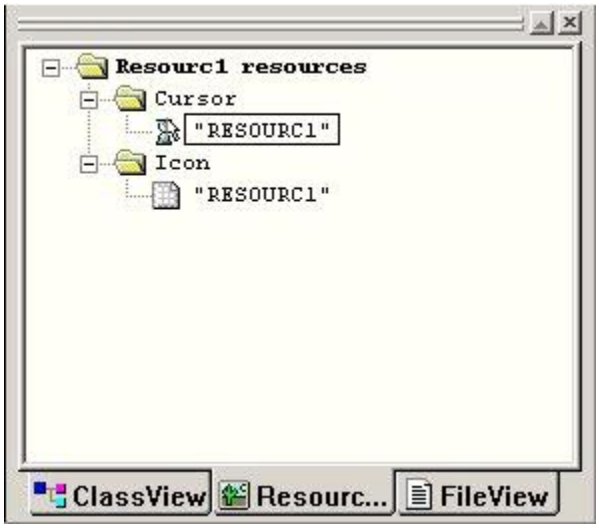
```
/*-----------------------------
   RESOURC1.RC resource script
  -----------------------------*/

resourc1  ICON    resourc1.ico
resourc1  CURSOR  resourc1.cur
```

ICON, CURSOR등은 리소스 스크립트의 키워드(keyword)이며, 선두의 resourc1은 명칭(identifier)이다. 그러므로 아이콘과 커서를 기술하는 스크립트 문법은 각각 다음과 같다.

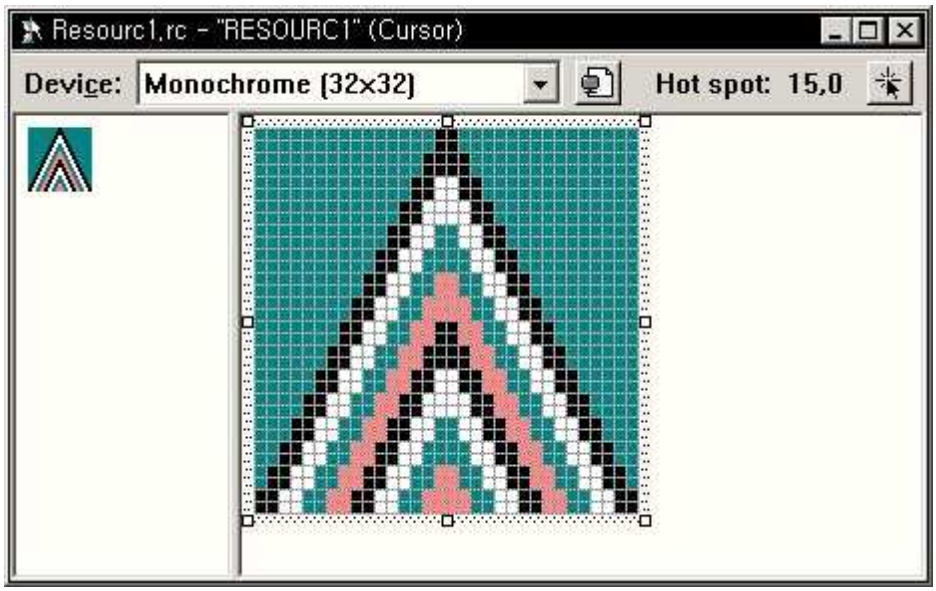
```
<identifier> ICON <filename>
<identifier> CURSOR <filename>
```

컴파일러가 고급화되기 전까지만 해도 이러한 리소스 스크립트를 만들 수 있는 유일한 도구는 텍스트 에디터(text editor)였지만, 비주얼 C++은 통합개발환경(IDE: integrated development environment)내에 비주얼 리소스 에디터를 포함하고 있어서, 쉽게 리소스를 편집하는 것이 가능하다. 리소스 편집기가 하는 일은 사용자가 그래피컬하게 편집한 리소스를 기술하는 스크립트 언어와 리소스 ID를 자동으로 생성하고 관련 헤더 파일을 생성하는 일이다. 위의 예를 비주얼 C++에서 읽어 들이면 아래 [그림 9.2]처럼 IDE의 리소스 뷰(resoruce view)에 트리 뷰(tree view) 컨트롤로 리소스들이 표시된다.



[그림 9.2] 리소스 뷰: 리소스 뷰에는 프로그램에서 사용하는 모든 리소스 들이 표시된다. 사용자는 편집을 원하는 리소스 ID를 선택하여 각각의 리소스 편집기를 실행할 수 있다. 예를 들면 사용자가 Cursor → "RESOURC1"을 선택하면 커서 에디터가 실행되어, 해당 커서 파일을 편집할 수 있다.

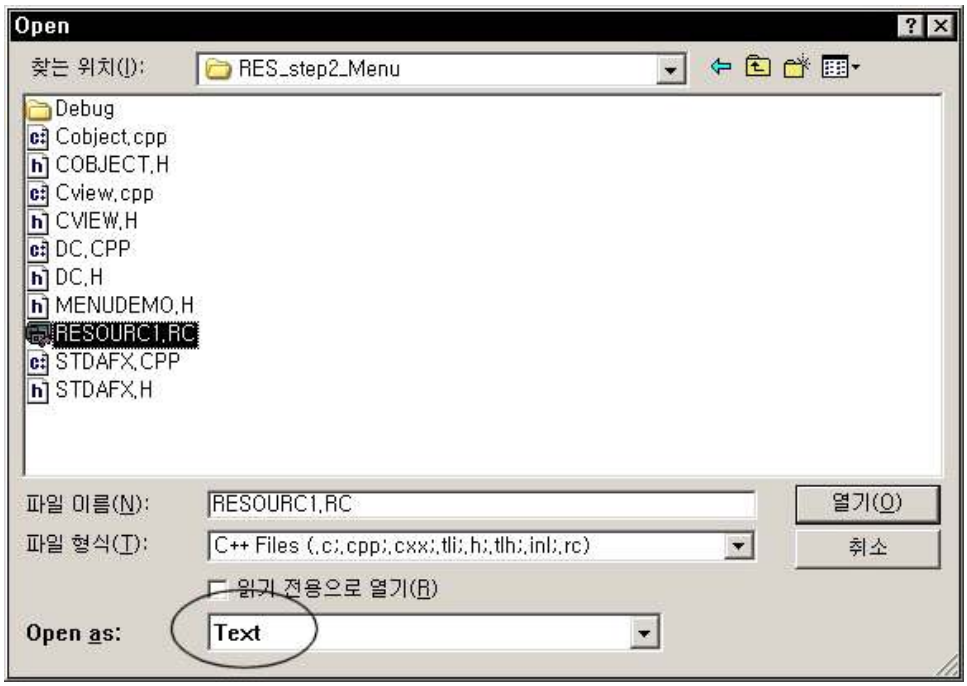
리소스 뷰에서 편집을 원하는 리소스를 선택하여 각 리소스 타입의 리소스 편집기를 실행할 수 있다. 예를 들면, Cursor→"RESOURC1"을 선택하면 커서 에디터가 실행되면서 "RESOURC1"이라는 명칭으로 지정된 커서 파일, resourc1.cur을 연다. 아래 [그림 9.3]을 보자.



[그림 9.3] 커서 에디터: 커서 에디터를 이용하여 응용 프로그램에서 사용하는 커서를 편집할 수 있다.

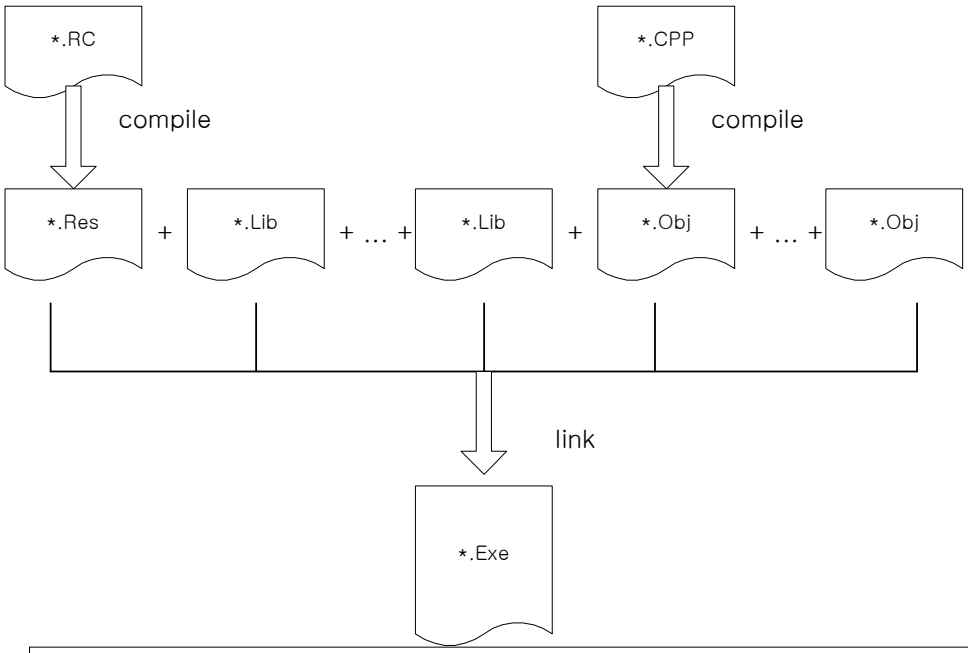
IDE의 리소스 에디터는 최종 편집 결과로 리소스 스크립트를 생성하는 일을 한다. 즉, 이전에는 텍스트 에디터로 했던 일들을 비주얼하게 편집하는 것이다. 원한다면 예전의 방식대로 텍스트 에디터를 사용하여 리소스 스크립트 파일(*.RC)를 직접 편집할 수도 있다. MFC 프로그래밍에서 그렇게 할 일은 자주 없겠지만, 리소스 스크립트를 직접 편집할 수 있는 능력도 반드시 필요하다□.

<저자한마디>
필자의 경험으로 볼 때, 기본적인 원리와 내부 동작을 이해하는 것은 재난 방지 훈련과 비슷하다. 평생을 거쳐서 한번도 사용되지 않을 수도 있지만, 훈련이 없다면 재난에 대비할 수 없다.
</저자한마디>



[그림 9.4] 리소스를 텍스트로 열기: 열기 대화상자에서 Open as 콤보 박스에서 Text를 선택해서 *.RC파일을 텍스트 파일로 연다.

MFC의 비주얼 리소스 에디터가 단지 *.rc만을 최종적으로 출력한다는 것을 염두에 두자. 실행 파일(*.exe)을 만들기 위해서 컴파일러는 소스 코드를 오브젝트 파일(object file, *.obj)로 만들고, 외부 함수들을 라이브러리 파일(*.lib)에서 불러와서 결합한다. 프로젝트에 명시한 리소스 스크립트가 있다면 컴파일러는 소스 코드 컴파일 이후 리소스 컴파일러를 실행한다. 리소스 컴파일러는 *.rc파일을 읽어 들여서 최종 실행 파일에 링크 가능한 이진 리소스(*.res)를 생성하고, 마지막 링크 단계에서 *.obj, *.lib, *.res를 결합하여 최종 실행파일 *.exe를 만드는 것이다. 아래 [그림 9.5]를 보자.



[그림 9.5] 리소스 컴파일: 최종 실행 파일을 만들기 위해서는 이진 리소스 파일(*.res), 오브젝트 파일(*.obj)과 라이브러리 파일(*.lib)이 필요하다. 리소스 에디터는 단지 *.rc 파일을 만드는 일만 한다.

리소스를 이용하는 첫 번째 프로그램은 응용 프로그램의 클라이언트 영역에 표시되는 마우스 커서를 바꾸고, 왼쪽 버튼을 눌렀을 때, 리소스에 지정된 아이콘을 클라이언트 영역에 출력하는 간단한 프로그램이다.

먼저 CObject의 InitInstance()를 수정한다. 윈도우를 생성하기 전 윈도우 클래스(window class)를 등록할 때, LoadIcon()과 LoadCursor()를 사용하여 리소스에 있는 리소스 핸들(resource handle)을 얻어서 윈도우 클래스에 설정한다. 각 함수의 두 번째 파라미터는 리소스 스크립트에 명시된 ID이다.

```
void CObject::InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                           int iCmdShow)
{
    //{{seojt
    hInst=hInstance;
    //}}seojt
    wndclass.cbSize         =sizeof(wndclass);
    wndclass.style          =CS_HREDRAW|CS_VREDRAW;
    wndclass.lpfnWndProc    =WndProc;
    wndclass.cbClsExtra     =0;
```

```
    wndclass.cbWndExtra     =0;
    wndclass.hInstance      =hInstance;
    //{{seojt
    wndclass.hIcon          =LoadIcon(hInstance,"resourc1");
    wndclass.hCursor        =LoadCursor(hInstance,"resourc1");
    //}}seojt
    wndclass.hbrBackground  =(HBRUSH)GetStockObject(WHITE_BRUSH);
    //wndclass.hbrBackground  =(HBRUSH)GetStockObject(NULL_BRUSH);
    wndclass.lpszMenuName   =NULL;
    wndclass.lpszClassName  =szAppName;
    //{{seojt
    wndclass.hIconSm        =LoadIcon(hInstance,"resourc1");
    //}}seojt

    RegisterClassEx(&wndclass);

    hwnd=CreateWindow(
        szAppName,            //window class name
        "Res step1: Icon",    //window caption
        //WS_POPUP,           //window style
        WS_OVERLAPPEDWINDOW,  //window style
        CW_USEDEFAULT,        //initial x position
        CW_USEDEFAULT,        //initial y position
        CW_USEDEFAULT,        //initial x size
        CW_USEDEFAULT,        //initial y size
        //300,300,
        NULL,                 //parent window handle
        NULL,                 //window menu handle
        hInstance,            //program instance handle
        NULL);                //creation parameters

    ShowWindow(hwnd,iCmdShow);
    UpdateWindow(hwnd);
}//CObject::InitInstance
```

이제 CView의 OnCreate()에서 클라이언트 영역에 아이콘을 그리기 위해 아이콘 핸들 hIcon을 저장해 놓는다. 그리고 GetSystemMetrics()를 호출해 아이콘의 크기를 구해 놓는다.

```
LRESULT CView::OnCreate(WPARAM wParam,LPARAM lParam)
{
    //{{seojt
```

```
    hIcon=::LoadIcon(hInst,"resourc1");//Is :: really necessary?
    cxIcon=::GetSystemMetrics(SM_CXICON);
    cyIcon=::GetSystemMetrics(SM_CYICON);
    //}}seojt
    return 0L;
}//CView::OnCreate
```

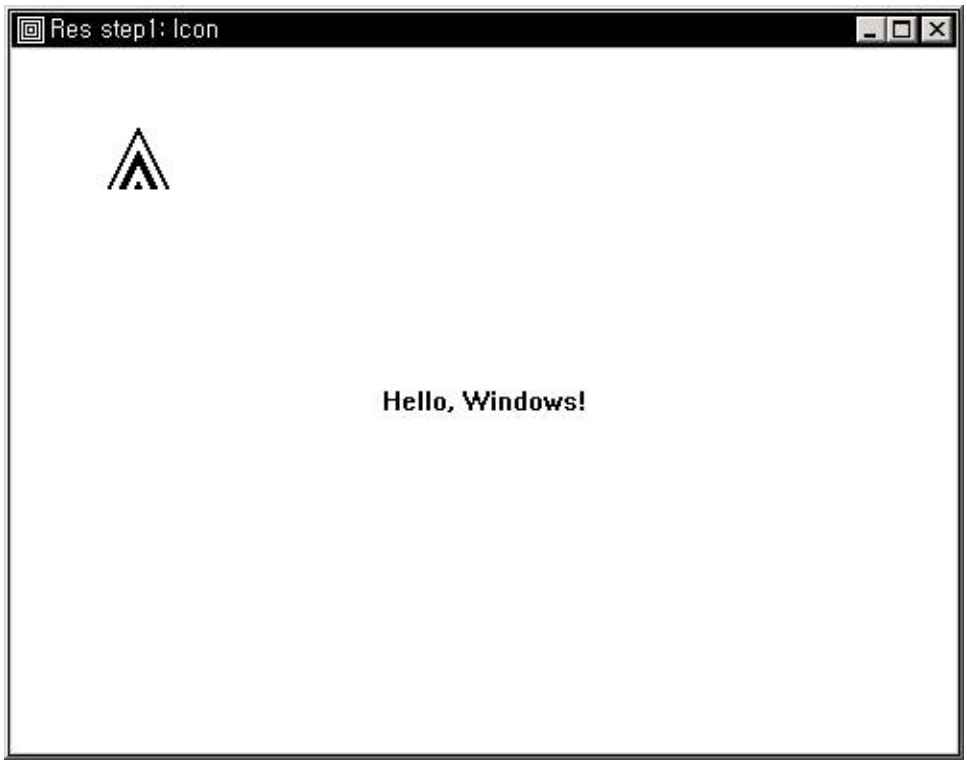
CView의 OnLButtonDown()에 아이콘을 그리는 루틴을 완성한다. CDC클래스에 추가적인 DrawIcon() 멤버 함수를 구현한다.

```
LRESULT CView::OnLButtonDown(WPARAM wParam,LPARAM lParam)
{
    CDC dc(this);
    //{{seojt
    int x,y;

    for (y = cyIcon ; y < cyClient ; y += 2 * cyIcon)
    {
        for (x = cxIcon ; x < cxClient ; x += 2 * cxIcon)
        {
            dc.DrawIcon (x, y, hIcon) ;
              //BOOL DrawIcon(HDC, int, int, HICON);
        }//for
    }//for
    //}}seojt

    return 0L;
}//CView::OnLButtonDown
```

DrawIcon()은 클라이언트 영역에 아이콘간격으로 공백을 두고 아이콘들을 꽉 차게 그린다. 프로그램의 실행 결과는 아래 [그림 9.6]과 같다.



[그림 9.6] 실행화면: 사용자가 지정한 커서가 클라이언트 영역에서 표시된다. 왼쪽 버튼을 누르면 사용자 아이콘을 클라이언트 영역에 그린다.

클라이언트 영역에 마우스 커서가 있을 때, 윈도우 클래스에서 지정한 커서가 표시된다. 왼쪽 버튼을 누르면 클라이언트 영역에 아이콘을 그린다.

예의 소스를 작성하는 과정에서 CObject의 멤버를 변경했다. CView를 제외한 나머지 전부가 MFC에 의해 이미 제공되는 코드라면 MFC의 소스를 직접 건드린 셈인데 이것은 바람직하지 않다. 어떻게 이러한 문제를 해결할 수 있을까?

MFC의 설계자들은 이 문제를 가상 함수(virtual function)로 해결했다. 즉 변경이 필요할 것으로 예상되는 모든 곳에 적절한 가상 함수를 만들어 이를 호출하도록 한 것이다.

윈도우를 생성하는 CreateWindow() 호출 이전에 윈도우의 스타일(style)과 위치 및 크기를 변경할 필요가 있다. 이 때를 대비해서 **PreCreateWindow**() 라는 가상 함수를 준비할 수 있다. CObject클래스가 PreCreateWindow()를 가

상함수로 가지도록 변경□한다. 또한 CObject의

<여기서 잠깐>
실제의 MFC 코드에서는 윈도우 클래스의 베이스 클래스로 사용되는 CWnd가 PreCreateWindow()를 가상함수로 가진다.
</여기서 잠깐>

InitInstance()는 CreateWindow()를 호출하기 전에 PreCreateWindow()를 호출하도록 변경될 것이다. 가상 함수의 이름은 On...()으로 시작하지 않는다. 왜냐하면 이벤트 핸들러가 아니기 때문이다. 하지만, 이벤트에 의해 호출되는 가상함수는 On...()으로 시작하므로 주의하도록 하자.

MFC에 사용된 이러한 가상 함수의 종류와 호출 시점 및 메시지 핸들러와의 상관 관계를 아는 것은 MFC 프로그래밍의 넘어야 할 산이다.

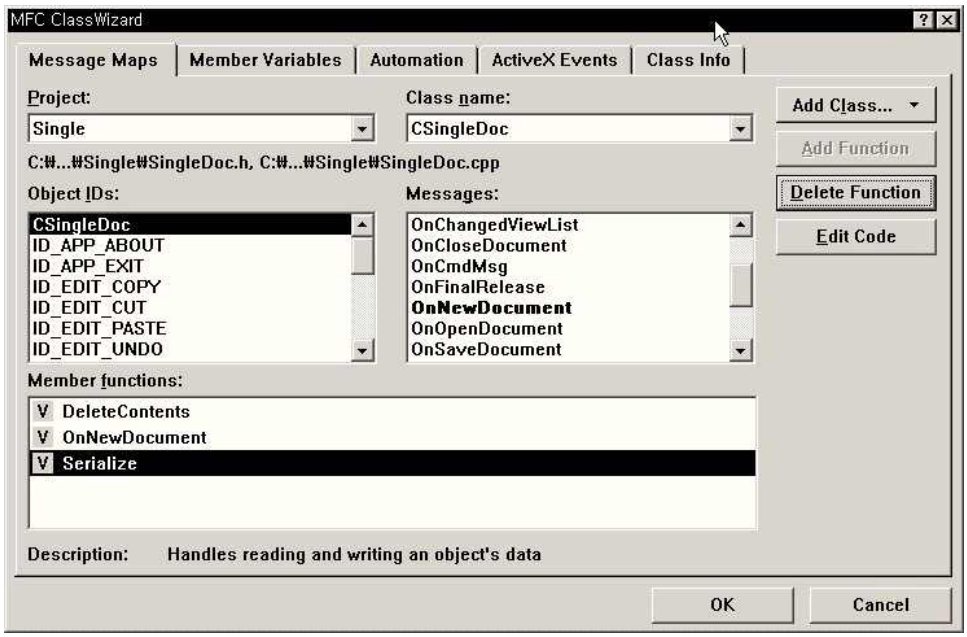
MFC의 가상함수들

<절도비라>
이 절에서는 Single 다큐먼트 프로젝트에 존재하는 MFC의 가상 함수들을 살펴본다. 리소스 관련 주제를 다루면서 MFC의 가상함수들을 살펴보는 이유는 무엇일까? 그것은 이 장의 주제가 특별한 MFC 클래스와 연관된 부분이 없기 때문이며 10장부터 다시 시작할 본격적인 MFC 코드 분석의 리듬을 유지하기 위해서이다.
</절도비라>

Single Document로 생성된 기본 코드에서 어떠한 가상 함수가 존재하는지 살펴보자. MFC 소스를 직접 확인해 볼 수도 있고, 클래스 위저드를 이용할 수도 있다□.

<여기서 잠깐>
가상함수가 어떻게 동작하는지 4장을 통해 다시 확인하기 바란다. 어떤 멤버 함수가 가상이라는 의미는 그 함수가 상속된 클래스에서 오버라이딩(overriding)을 통해 확장되거나 재정의할 수 있다란 의미이다.
</여기서 잠깐>

7장에서 작성했던 Single 프로젝트를 연다. 그리고, 다큐먼트 클래스의 가상 함수들을 살펴보자.



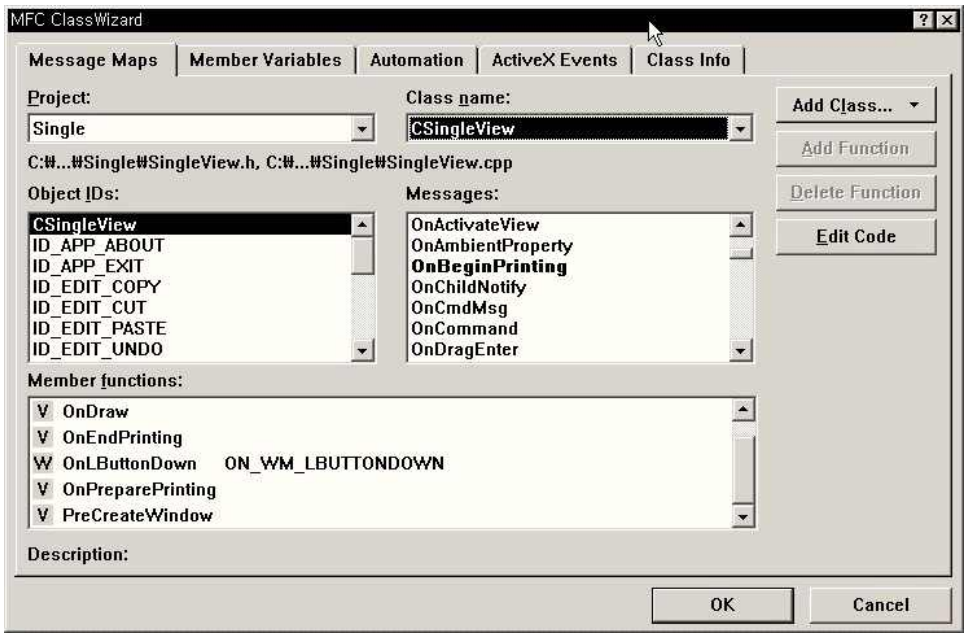
[그림 9.7] CSingleDoc의 가상 함수들:

[그림 9.7]의 Member functions 리스트에는 이미 생성된 가상 함수들을 보여준다. 가상함수는 함수 이름 앞에 대문자 V가 표시되어 있다. 그러한 가상 함수에는

```
DeleteContents();
OnNewDocument();
Serialize();
```

가 있음을 확인할 수 있다. 함수들은 이름이 의미하듯이 다큐먼트의 내용을 지울 때, 새 다큐먼트를 만들 때, 직렬화할 때 호출된다. 파일→새 파일을 선택하면 DeleteContents()와 OnNewDocument() 중 어느 것이 먼저 호출될까? 독자들이 꼭 브레이크 포인트를 걸고 모든 가상 함수들의 호출 시점을 추적(trace)해 보기 바란다.

이제 Class name 콤보 박스로 [그림 9.8]처럼 CSingleView를 선택한다.



[그림 9.8] CSingleView의 가상함수들:

Member functions 리스트에서 확인할 수 있는, 뷰의 가상 함수는

```
OnBeginPrinting();
OnDraw();
OnEndPrinting();
OnPreparePrinting();
PreCreateWindow();
```

이다. OnDraw()는 WM_PAINT메시지가 발생했을 때 OnPaint() 핸들러가 호출한다□. PreCreateWindow()는 WM_CREATE 핸들러 즉 OnCreate()가 호출

<저자 한마디>

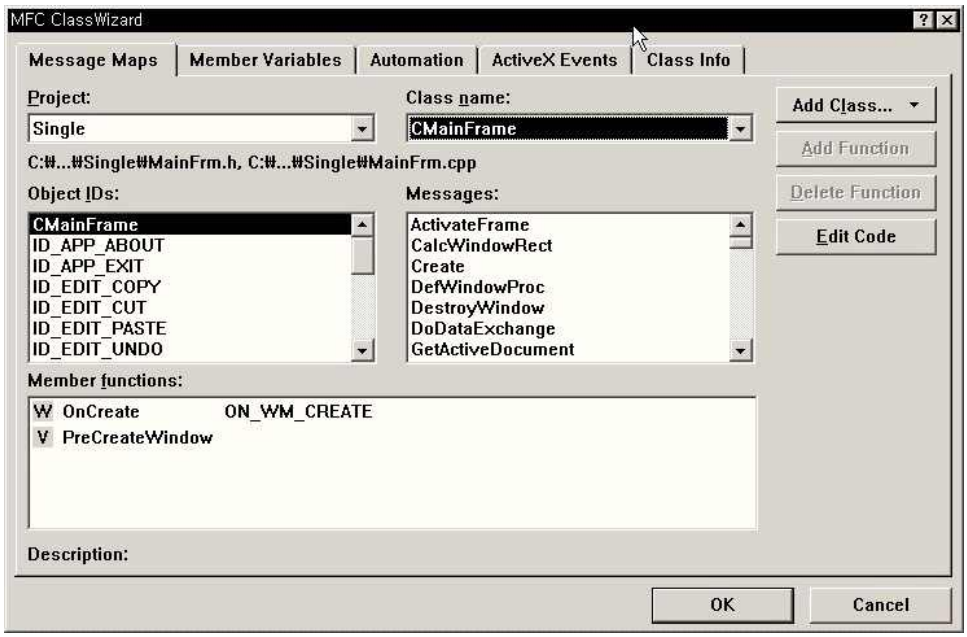
OnDraw()는 반드시 존재해야 한다. 심지어 OnPaint()에서 OnDraw()를 주석 처리해서 호출을 취소하는 것도 불가능하다. 왜 그런가?

</저자 한마디>

되기 전 호출된다. 윈도우의 초기 크기나 위치를 변경시키기 위해

PreCreateWindow()를 오버라이드한다.

이제 [그림 9.9]에서 처럼 Class name에서 CMainFrame을 선택한다.



[그림 9.9] CMainFrame의 가상함수들:

CMainFrame에는 WM_CREATE 핸들러 OnCreate()가 이미 작성되어 있음을 확인할 수 있다. 메인 프레임에도 PreCreateWindow()가 가상 함수로 존재한다. 이제 뷰에 WM_CREATE를 매핑하고, 뷰와 메인 프레임의 OnCreate()와 PreCreateWindow()에 모두 브레이크 포인트를 걸고, 이들의 호출 순서를 확인해 두기 바란다.

이들의 호출 순서는

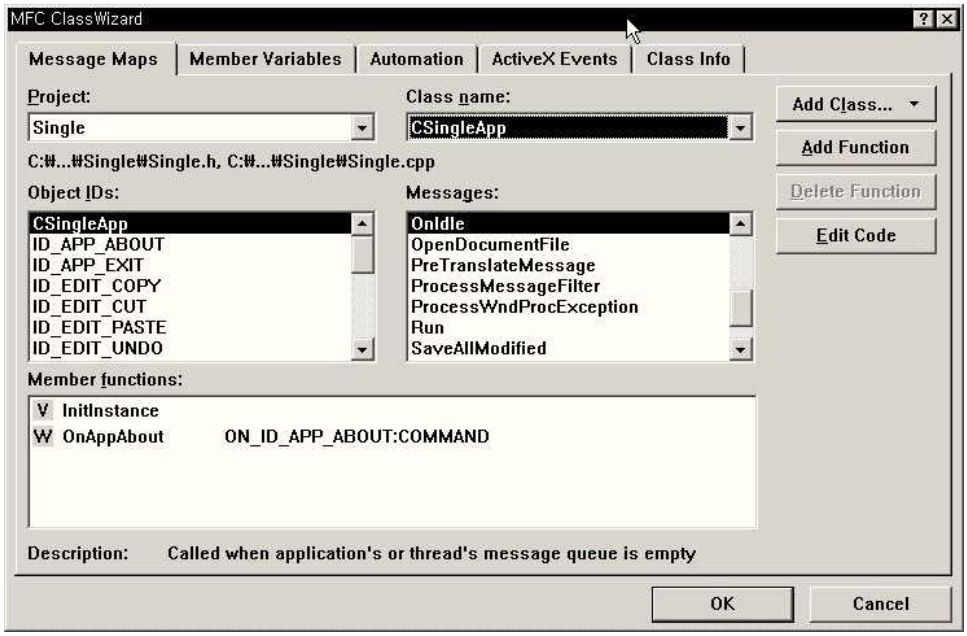
```
(1) CMainFrame::PreCreateWindow();
(2) CMainFrame::OnCreate();
(3) CSingleView::PreCreateWindow();
(4) CSinglrView::OnCreate();
```

이다. 상태바와 툴바는 언제 생성될까? 그것은 CMainFrame의 OnCreate()에서 한다. 상태바와 툴바는 메인 프레임의 멤버이므로 이것은 당연하다.

우리는 여기서 메인 프레임의 클라이언트 영역을 얻는 것은 대부분의 경우

무의미하다는 것을 알 수 있다. 그 영역은 툴바와 상태바를 포함하므로, 메인 프레임의 클라이언트 영역에 그리기를 할 때는 툴바와 상태바의 위치를 고려해야 한다. 뷰는 메인 프레임의 클라이언트 영역이 아니라, 메인 프레임의 클라이언트 영역 중 상태바와 툴바를 제외한 부분을 덮는 윈도우이다.

이제 CSingleApp 클래스의 가상 함수를 보자.



[그림 9.10] CSingleApp의 가상 함수들:

InitInstance()는 우리가 했던 바로 그 일을 한다! 원한다면 아래의 함수들을 필요에 따라 오버라이드 할 수 있다.

```
Run()
ExitInstance()
OnIdle()
```

이 함수들도 우리가 했던 바로 그 일을 한다.

CView의 가상 함수를 모두 살펴보자. 아래의 소스는 MFC의 CView에서 가상 함수와 미리 정의된 메시지 맵 부분만을 남겨둔 것이다. CView를 보면

대부분의 멤버가 가상 함수인 것을 알 수 있는데, 그도 그럴 것이 CView는 추상 베이스 클래스(abstract base class)□이기 때문이다.

<여기서 잠깐>

순수 가상 함수를 한 개 이상 가지는 클래스는 객체를 만들 수 없고, 반드시 다른 클래스의 베이스 클래스로 사용된다. 이러한 클래스를 추상 클래스라 한다. OnDraw()는 순수 가상 함수이다.

</여기서 잠깐>

```
class AFX_NOVTABLE CView : public CWnd
{
    DECLARE_DYNAMIC(CView)
public:
    virtual BOOL IsSelected(const CObject* pDocItem) const;
    // support for OLE

    // OLE scrolling support (used for drag/drop as well) // (1)
    virtual BOOL OnScroll(UINT nScrollCode, UINT nPos,
        BOOL bDoScroll = TRUE);
    virtual BOOL OnScrollBy(CSize sizeScroll, BOOL bDoScroll = TRUE);

    // OLE drag/drop support
    virtual DROPEFFECT OnDragEnter(COleDataObject* pDataObject,
        DWORD dwKeyState, CPoint point);
    virtual DROPEFFECT OnDragOver(COleDataObject* pDataObject,
        DWORD dwKeyState, CPoint point);
    virtual void OnDragLeave();
    virtual BOOL OnDrop(COleDataObject* pDataObject,
        DROPEFFECT dropEffect, CPoint point);
    virtual DROPEFFECT OnDropEx(COleDataObject* pDataObject,
        DROPEFFECT dropDefault, DROPEFFECT dropList, CPoint point);
    virtual DROPEFFECT OnDragScroll(DWORD dwKeyState, CPoint point);

    virtual void OnPrepareDC(CDC* pDC, CPrintInfo* pInfo = NULL);

    virtual void OnInitialUpdate();
    // called first time after construct // (2)

protected:
    // Activation
```

```
    virtual void OnActivateView(BOOL bActivate, CView* pActivateView,
                                CView* pDeactiveView);
    virtual void OnActivateFrame(UINT nState, CFrameWnd* pFrameWnd);

    // General drawing/updating // (3)
    virtual void OnUpdate(CView* pSender, LPARAM lHint,
                          CObject* pHint);
    virtual void OnDraw(CDC* pDC) = 0;

    // Printing support // (4)
    virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        // must override to enable printing and print preview

    virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnPrint(CDC* pDC, CPrintInfo* pInfo);
    virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

    // Advanced: end print preview mode, move to point
    virtual void OnEndPrintPreview(CDC* pDC, CPrintInfo* pInfo,
        POINT point, CPreviewView* pView);

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

    virtual void CalcWindowRect(LPRECT lpClientRect,
        UINT nAdjustType = adjustBorder);
    virtual CScrollBar* GetScrollBarCtrl(int nBar) const;
public:
    virtual BOOL OnCmdMsg(UINT nID, int nCode, void* pExtra,
        AFX_CMDHANDLERINFO* pHandlerInfo);
protected:
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    virtual void PostNcDestroy();

    //{{AFX_MSG(CView) // (5)
    afx_msg int OnCreate(LPCREATESTRUCT lpcs);
    afx_msg void OnDestroy();
    afx_msg void OnPaint();
```

```
    afx_msg int OnMouseActivate(CWnd* pDesktopWnd,
        UINT nHitTest, UINT message);
    // commands
    afx_msg void OnUpdateSplitCmd(CCmdUI* pCmdUI);
    afx_msg BOOL OnSplitCmd(UINT nID);
    afx_msg void OnUpdateNextPaneMenu(CCmdUI* pCmdUI);
    afx_msg BOOL OnNextPaneCmd(UINT nID);

    // not mapped commands - must be mapped in derived class
    afx_msg void OnFilePrint();
    afx_msg void OnFilePrintPreview();
    //}}AFX_MSG
};
```

소스에서 drag & drop과 관계된 가상 함수들(1), OnCrete() 이후에 단 한번 호출되는 함수인(2) OnInitialUpdate(), 다큐먼트의 UpdateAllView()에 대응하는 핸들러(3), 프린팅과 관계된 가상 함수들(4), 미리 정의된 윈도우 메시지 핸들러(5)를 확인할 수 있다. 아래의 문제는 독자들이 스스로 답해보기 바란다.

"OnCreate()도 어차피 한 번 호출된다. 그렇다면, 왜 OnInitialUpdate()가 필요한가?"

두 번째 예제: 메뉴의 추가

<절도비라>
아이콘 리소스를 추가한 이 장의 처음 프로그램에 메뉴 리소스를 추가하고, 메뉴 명령의 핸들러를 작성하는 방법을 익힌다. 덧붙여 메뉴 리소스가 추가되는 경우 변경되는 소스를 살펴본다.
</절도비라>

두 번째 프로젝트 ResStep2는 메뉴 리소스를 가지는 간단한 프로그램이다. 첫 번째 예에 메뉴를 추가하고, 뷰에 메뉴 명령의 핸들러 OnCommand()를 설치할 것이다. [예제 9.1]에 리소스 스크립트를 리스트하였다.

[예제 9.1] Resource1.rc

```
/*-----------------------------
   RESOURC1.RC resource script
  -----------------------------*/

resourc1  ICON    resourc1.ico
resourc1  CURSOR  resourc1.cur

#include "menudemo.h"

MenuDemo MENU // (1)
     {
     POPUP "&File" // (2)
          {
          MENUITEM "&New",                    IDM_NEW // (3)
          MENUITEM "&Open...",                IDM_OPEN // (4)
          MENUITEM "&Save",                   IDM_SAVE
          MENUITEM "Save &As...",             IDM_SAVEAS
          MENUITEM SEPARATOR // (5)
          MENUITEM "E&xit",                   IDM_EXIT
          }
     POPUP "&Edit"
          {
          MENUITEM "&Undo",                   IDM_UNDO
          MENUITEM SEPARATOR
          MENUITEM "Cu&t",                    IDM_CUT
          MENUITEM "&Copy",                   IDM_COPY
          MENUITEM "&Paste",                  IDM_PASTE
          MENUITEM "De&lete",                 IDM_DEL
          }
     POPUP "&Background"
          {
          MENUITEM "&White",                  IDM_WHITE, CHECKED
          MENUITEM "&Lt Gray",                IDM_LTGRAY
          MENUITEM "&Gray",                   IDM_GRAY
          MENUITEM "&Dk Gray",                IDM_DKGRAY
          MENUITEM "&Black",                  IDM_BLACK
          }
     POPUP "&Timer"
          {
          MENUITEM "&Start"                   IDM_START
          MENUITEM "S&top"                    IDM_STOP,  GRAYED
```

```
        }
    POPUP "&Help"
        {
        MENUITEM "&Help...",                IDM_HELP
        MENUITEM "&About MenuDemo...",      IDM_ABOUT
        }
    }
```

메뉴 리소스의 기술은

```
<메뉴이름> MENU
```

로 시작하고(1) 블록이 이어진다. 메뉴를 구성하는 각 팝업 메뉴는 메뉴 블록 안에 POPUP 블록으로 기술한다(2).

```
POPUP "팝업 메뉴 제목"
{
}
```

"팝업 메뉴 제목"은 액세스 키(access key) - Alt키와 같이 누르는 키 - 를 지정하기 위해 이스케이프 문자(escape character)로 &를 사용한다. 즉 &File은 F를 액세스 키로 등록하며, 화면에는

```
File
```

로 표시된다. &File처럼 표시하기 위해서는 &&File로 제목을 지정하며, 이 경우 등록되는 액세스 키는 없다. 팝업 메뉴의 각 메뉴 항목은

```
MENUITEM, "메뉴이름"[, 메뉴 ID]
```

로 지정한다. "메뉴 이름"도 &를 액세스 키 지정을 위한 이스케이프 문자로 사용한다. 메뉴 ID 등 리소스의 ID들은 일반적으로 별도의 헤더 파일 - 여기서는 menudemo.h - 에 매크로로 정의하여 사용한다(3). 메뉴 이름을 정할 때의 관례는 메뉴 선택의 동작이 추가적인 선택을 요구하면 끝에

```
...
```

을 붙이는 것이다(4). 예를 들면 Save 항목은 파일을 즉시 저장하므로

```
Save...
```

처럼 사용하면 안 된다. 하지만, Open 항목은 Open선택 후, 대화상자에서 파일을 선택하는 추가적인 작업이 있으므로,

```
Open...
```

처럼 항목의 이름을 정해야 한다. 메뉴 항목의 이름이

```
SEPARATOR
```

이면 메뉴 항목 자리에는 가로선이 표시된다. 이것은 메뉴 항목간의 그룹핑(grouping)을 위해 사용한다. 첫 번째 팝업 항목

```
POPUP "&File"
```

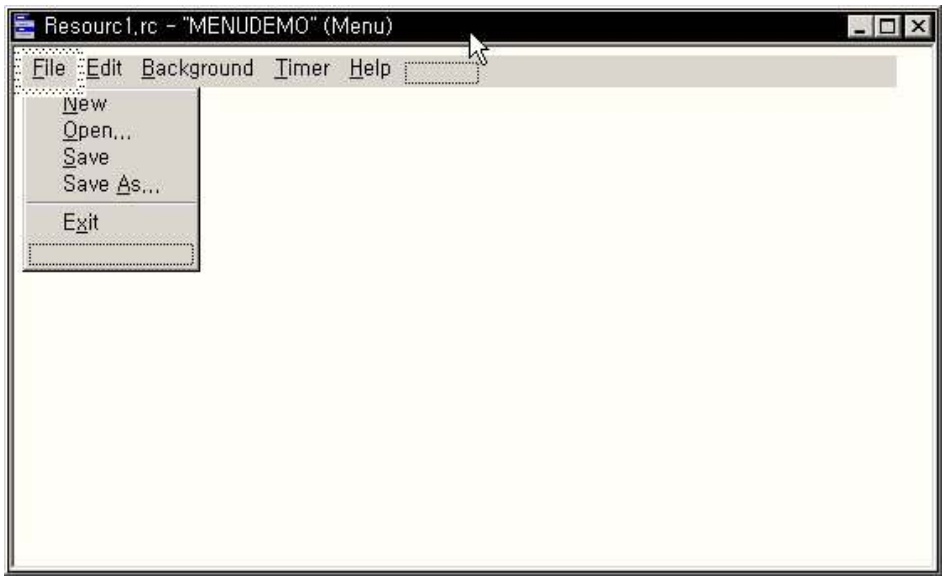
은 아래 [그림 9.11]의 팝업 메뉴를 기술한다.

[그림 9.11] POPUP "&File" 팝업 메뉴: 모두 6개의 항목이면, 추가적인 선택을 요구하는 항목의 끝에는 ...을 표시한다. 또한, 일반적으로 사용하는 액세스 키의 관례를 따라야 한다. 예를 들면 Exit의 경우 &Exit처럼 액세스 키를 등록하는 것은 좋지 않다.

이제 스크립트를 비주얼 C++에서 열어 보자. 리소스 뷰에서

```
Menu→"MENUDEMO"
```

를 선택하면 아래 [그림 9.12]처럼 메뉴 에디터가 실행된다.



[그림 9.12] 메뉴 에디터: 메뉴 에디터는 비주얼하게 편집한 메뉴의 레이아웃을 리소스 스크립트에 저장한다.

메뉴 에디터가 하는 일은 메뉴 스크립트를 수동으로 편집하는 대신, 비주얼한 편집의 결과를 리소스 스크립트의 메뉴 리소스에 추가하는 일이다. 메뉴 리소스는 윈도우 클래스를 등록할 때, 지정해 주어야 하고 이것은 CObject::InitInstance()에서 한다. 아래의 소스를 보자.

```
void CObject::InitInstance(HINSTANCE hInstance,PSTR szCmdLine,
                           int iCmdShow)
{
    hInst=hInstance;

    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(hInstance,"resourc1");
    wndclass.hCursor        =LoadCursor(hInstance,"resourc1");
    wndclass.hbrBackground  =(HBRUSH)GetStockObject(WHITE_BRUSH);
```

```
    //{{seojt
    wndclass.lpszMenuName   ="MenuDemo"; // (1)
    //}}seojt
    wndclass.lpszClassName  =szAppName;
    wndclass.hIconSm        =LoadIcon(hInstance,"resourc1");

    RegisterClassEx(&wndclass);

    hwnd=CreateWindow(
        szAppName,                  //window class name
        "Res step2: Menu",          //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
        hInstance,                  //program instance handle
        NULL);                      //creation parameters

    ShowWindow(hwnd,iCmdShow);
    UpdateWindow(hwnd);
}//CObject::InitInstance
```

응용 프로그램의 메인 메뉴로 리소스의 메뉴를 등록하기 위해서, 윈도우 클래스를 등록할 때, 메뉴 이름에 리소스 스크립트에 지정된 메뉴 이름을 지정해야 한다(1). 이제 응용 프로그램은 메뉴를 가지며, 클라이언트 영역은 프레임과 메뉴를 제외한 영역을 가리킨다. 그리고 뷰에 메뉴 핸들러를 작성해 보자. 메뉴 선택은

```
WM_COMMAND
```

메시지를 발생시킨다. WM_COMMAND 메시지는 또한 컨트롤, 액셀러레이터(accelerator)에 의해서도 발생할 수 있는데, 각각은 wParam의 상위 2바이트로 구분한다.

```
컨트롤의 통지(notification) 메시지: > 1
액셀러레이터: 1
메뉴: 0
```

이다. 그러므로 여러 상황에 의해 WM_COMMAND가 발생할 수 있는 경우, 반드시 wParam의 상위 워드(word)를 검사해 보아야 한다. 본 예에서는 컨트롤 혹은 액셀러레이터는 없으므로 바로 메뉴 ID - wParam의 하위 워드 - 만을 검사한다.

```
BEGIN_MESSAGE_MAP(CView)
    {WM_CREATE,CView::OnCreate},
    {WM_PAINT,CView::OnDraw},
    {WM_DESTROY,CView::OnDestroy},
    {WM_SIZE,CView::OnSize},
    {WM_LBUTTONDOWN,CView::OnLButtonDown},
    //{{seojt
    {WM_COMMAND,CView::OnCommand},
    //}}seojt
END_MESSAGE_MAP()
```

뷰에 OnCommand()의 선언을 추가하고, 구현 파일의 메시지 맵에 엔트리를 추가한다. 그리고 OnCommand() 핸들러를 아래와 같이 작성한다.

```
LRESULT CView::OnCommand(WPARAM wParam,LPARAM lParam) {
    HMENU hMenu;

    hMenu=GetMenu(hwnd);
    switch (LOWORD(wParam)) {
    case IDM_EXIT:
        SendMessage(hwnd,WM_CLOSE,0,0L);
        break;
    case IDM_WHITE:
        //qff: Question?
        //InvalidateRect(NULL, NULL, TRUE);
        SendMessage(hwnd, WM_LBUTTONDOWN, 0, 0);
        break;
    }//switch
    return 0L;
}//CView::OnCommand
```

OnCommand()가 하는 일은 File→Exit을 선택한 경우, WM_CLOSE 메시지를 전송하고, Background→White를 선택한 경우, WM_LBUTTONDOWN을 전송한다. 프로그램의 실행 결과는 아래 [그림 9.13]과 같다.

[그림 9.13] 실행화면: 메뉴의 Background→White를 선택하면 클라이언트 영역에 커서를 그린다. 또한 File→Exit을 선택하면 프로그램은 종료한다.

리소스 ID의 관리

<절도비라>

(리소스 ID를 관리하는 것이 중요하기 때문에 살펴보는 것이죠? 그 점을 부각시키고, 이를 이해해야 자신이 직접 텍스트 에디터로 리소스 크립트를 편집할 수 있다고 피력해주셔야 합니다. 다음 문장은 너무 순차적이고, 수동적으로 만드는 설명입니다.)

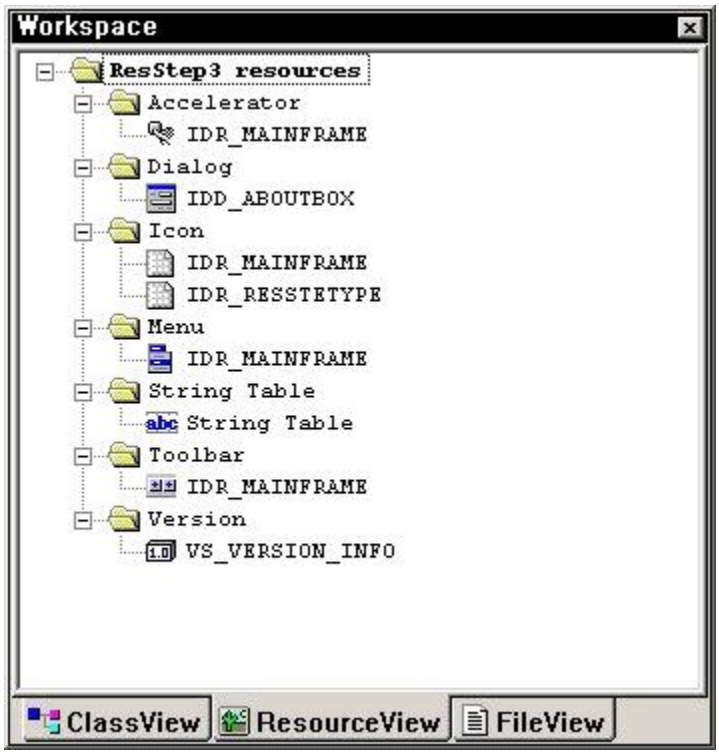
비주얼 C++의 리소스 에디터를 사용하는 경우, 리소스의 ID들은 자동으로 관리된다. 하지만, 프로젝트를 진행하다 보면, 리소스 ID를 직접 지정하거나, 편집해야하는 경우가 발생한다. 그러기 위해서는 비주얼 C++이 리소스 에디터를 이용하여 리소스를 관리하는 방법을 이해하고, 텍스트 에디터로 리소스 스크립트를 직접 편집하는 방법을 익히는 것이 필요하다.

</절도비라>

비주얼 C++의 리소스 에디터와 관련하여 알아 둘 사항은 리소스의 ID를 헤더 파일에서 관리하는 방법이다. 비주얼 C++은 리소스의 ID들을 아래의 파일에 관리한다.

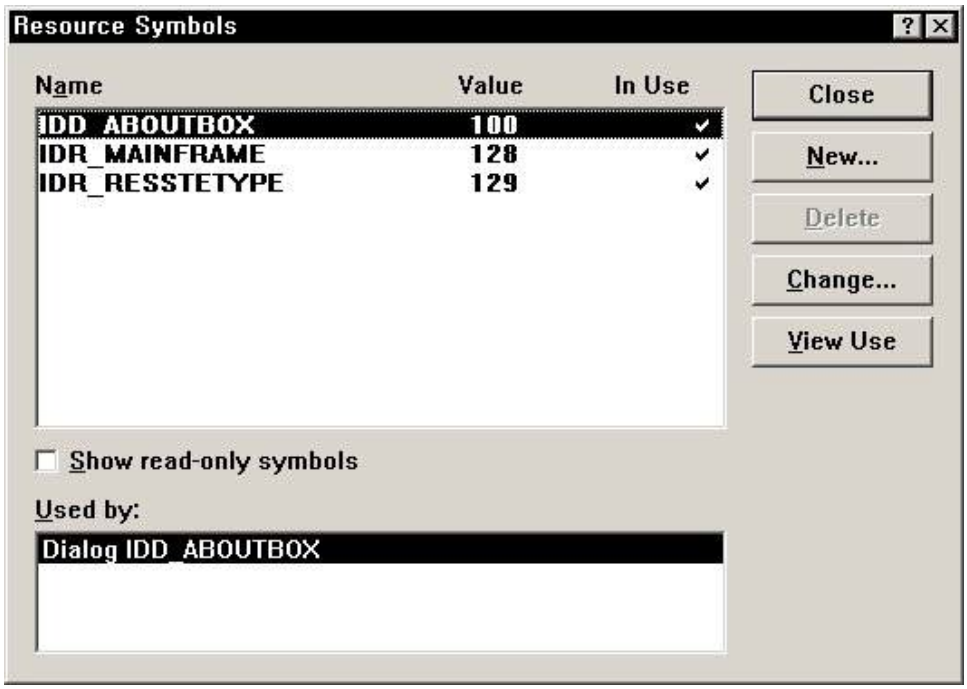
resource.h

프로젝트 이름을 ResStep3으로 하여 MFC 싱글 다큐먼트 응용 프로그램을 만들어 보자. 그러면 프로젝트에 추가된 디폴트 리소스는 아래 [그림 9.14]와 같다.



[그림 9.14] 디폴트 리소스: 싱글 다큐먼트의 디폴트 리소스에 많은 리소스가 포함된다.

이제 View→Resource Symbols...를 선택하여 resource.h에 정의된 리소스 ID를 확인해 보자.



[그림 9.15] 리소스 심벌: Resource Symbols...를 선택하면 resource.h에 정의된 리소스 ID들을 확인해 볼 수 있다.

리소스 이름의 실제 값이 100, 128과 129인 것을 확인하고 resource.h 파일을 열어 보자. 파일의 내용은 [예제 9.2]와 같다.

[예제 9.2] Resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by RESSTEP3.RC
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_RESSTETYPE                  129

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
```

```
#define _APS_3D_CONTROLS                1
#define _APS_NEXT_RESOURCE_VALUE        130
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           101
#define _APS_NEXT_COMMAND_VALUE         32771
#endif
#endif
```

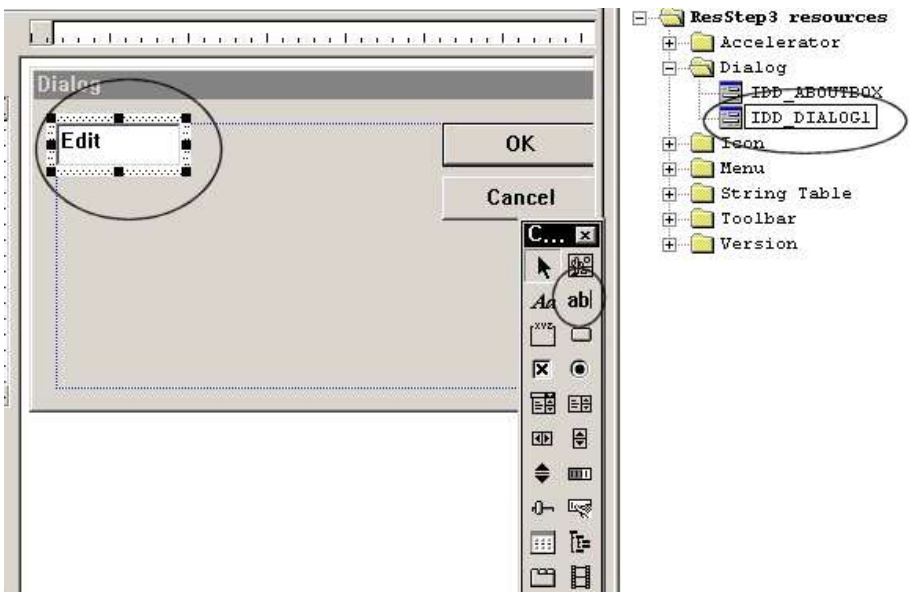
헤더 파일을 보면 특별하게 정의된 아래 정의를 볼 수 있다.

```
APSTUDIO_INVOKED
```

그 중 _APS_NEXT_RESOURCE_VALUE의 값이 130인데 이것은 다음에 추가되는 리소스의 ID를 130으로 사용하겠다는 의미이다. 즉 리소스 에디터는 resource.h에 정의된 APSTUDIO_INVOKED를 찾아서 리소스의 ID들을 관리하는 것이다.

새로 추가되는 컨트롤은 1000부터 시작할 것이고, 메뉴 명령은 32771부터 시작할 것이다.

이제 리소스 뷰에서 대화상자를 추가하고, 대화상자에 텍스트 박스를 하나 추가해 보자.



[그림 9.16] 대화상자와 컨트롤의 추가: 대화 상자와 컨트롤을 추가하면 리소스 편집기는 resource.h를 자동으로 갱신한다.

그러면 View→Resource Symbols... 혹은 resource.h 파일을 열어서 변경된 사항을 확인해 볼 수 있다. 변경된 resource.h 파일은 [예제 9.3]과 같다.

[예제 9.3] 변경된 Resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by ResStep3.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_RESSTETYPE                  129
#define IDD_DIALOG1                     130
#define IDC_EDIT1                       1000

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                1
#define _APS_NEXT_RESOURCE_VALUE        131
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1001
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

리소스 편집기가 ID를 관리하는 방법은 알아 둘 필요가 있다. 이것은 서로 다르게 작성된 두 프로젝트에서 리소스를 합치는 과정에서 수동으로 resource.h를 편집할 때 필요하다.

요약

이 장에서 비주얼 C++ 의 비주얼 리소스 에디터가 리소스들을 관리하는 방법에 대해서 살펴보았다. 두 개의 프로젝트를 하나로 합치는 경우가 발생한다면 리소스 스크립트를 직접 수정해야 하므로 원리를 이해하는 것이 필요하다.

- **리소스 스크립트**는 윈도우즈 응용 프로그램이 사용하는 리소스를 기술하는 특별한 언어이다.
- 비주얼 C++ 6의 **비주얼 리소스 에디터**를 사용하면 직접 리소스 스크립트를 편집할 일은 거의 없지만, 특별한 경우를 대비하여 원리를 이해해 두는 것이 필요하다.
- **리소스 컴파일러**에 의해 컴파일된 바이너리 리소스(.res)는 링크 과정에서 실행파일에 합쳐진다.
- MFC 응용 프로그램은 다수의 **가상 함수**를 가지며, 이들의 호출 시점 및 동작을 이해하는 것이 필요하다.
- 비주얼 리소스 에디터는 리소스 ID를 자동으로 관리하기 위해 리소스 편집 시에 **APSTUDIO_INVOKED**을 자동으로 define한다. 리소스 헤더 파일은 이 매크로를 이용하여 다음 ID를 자동으로 관리한다.

[문서의 끝]