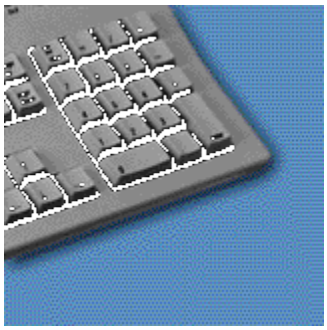


8. CClientDC의 원리 분석

<장도비라>
이 장에서 우리는 7장에서 시작한 Single 프로젝트의 CSingleView 클래스의 OnDraw()를 분석하는데 집중한다. OnDraw()는 CDC객체에 대한 포인터를 파라미터로 받는데, 이 장에서 우리는 GDI함수의 래퍼 클래스인 CDC의 원리를 이해하고, CDC를 상속받아 구현한 CClientDC와 CPaintDC의 차이점을 살펴볼 것이다. 먼저 간단한 GDI 응용 프로그램을 C++로 구현할 것이고, 이것을 CDC를 사용하는 버전으로 고쳐나갈 것이다. 또한 똑같이 동작하는 MFC 코드를 작성해 보고, MFC에 구현된 CClientDC를 살펴본다.
</장도비라>

이 장에서 GDI(graphic device interface)의 래퍼 클래스(wrapper class)인 CPaintDC와 CClientDC 등의 원리에 대해서 알아본다. 살펴볼 내용은 아래와 같다.

- GDI의 래퍼 클래스인 CDC의 원리
- CDC를 상속 받아 구현한 CClientDC의 원리
- CClientDC와 CPaintDC의 차이점
- 가상함수 OnDraw()가 동작하는 원리

우리는 먼저 순수한 GDI API를 이용하는 프로그램을 설계할 것이다. 다음으로 GDI의 래퍼 클래스인 CDC를 만들어, 순수한 GDI API를 이용하는 프로그램을 변경시킬 것이다. 마지막으로 이러한 원리가 그대로 MFC에 적용되고 있음을 살펴볼 것이다.

이제 마우스 왼쪽 버튼을 눌렀을 때, 클라이언트 영역을 채우는 사인 곡선(sine curve)을 그리는 GDI 프로그램을 구현해 보자.

단계 1: GetDC()의 이용

<절도비라>
이 절에서는 API 함수 GetDC()와 PolyLine()을 이용해 사인 곡선을 그리는 응용 프로그램을 작성한다. WM_SIZE와 WM_LBUTTONDOWN 메시지를 매핑하고, WM_LBUTTONDOWN 메시지의 핸들러에서 사인 곡선을 그린다. 다음 단계에서 이 프로그램을 CDC 클래스를 사용하는 프로그램으로 변환할 것이다.
</절도비라>

먼저 사인 곡선을 구성하는 점의 개수와 360도의 라디안(radian)을 정의한다.

```
#define NUM    1000
#define TWOPI  (2 * 3.14159)
```

그리고, 클라이언트 영역의 크기(1)와 사인 곡선을 구성하는 점들의 위치를 담을 변수를 CView의 멤버로 선언한다(2).

```
class CView : public CObject
{
public:
    // ...
    POINT pt[NUM]; // (2)
    int cyClient;  // (1)
    int cxClient;

    // ...

    LRESULT OnSize(WPARAM wParam,LPARAM lParam); // (3)
    LRESULT OnLButtonDown(WPARAM wParam,LPARAM lParam); // (4)
    // ...
};//class CView
```

또한, CView에 추가적인 두개의 메시지 핸들러를 구현한다. 하나는 윈도우의 크기가 변할 때마다, cxClient값을 갱신하기 위한 WM_SIZE이고(3), 다른

하나는 그림을 그리기 위한 WM_LBUTTONDOWN메시지이다(4).

헤더 파일에 추가된 핸들러 선언에 대응하는 구현 부분을 구현 파일 CView.cpp에 해 주어야 한다. 먼저 메시지 맵에 엔트리를 추가한다(1).

```
BEGIN_MESSAGE_MAP(CView)
    {WM_CREATE,CView::OnCreate},
    {WM_PAINT,CView::OnDraw},
    {WM_DESTROY,CView::OnDestroy},
    //{{seojt
    {WM_SIZE,CView::OnSize}, // (1)
    {WM_LBUTTONDOWN,CView::OnLButtonDown},
    //}}seojt
END_MESSAGE_MAP()
```

그리고, 핸들러를 구현한다. OnSize()는 클라이언트의 크기를 저장해 둔다. LOWORD()와 HIWORD()는 각각 4바이트 정수의 하위 워드와 상위 워드를 구하는 매크로이다.

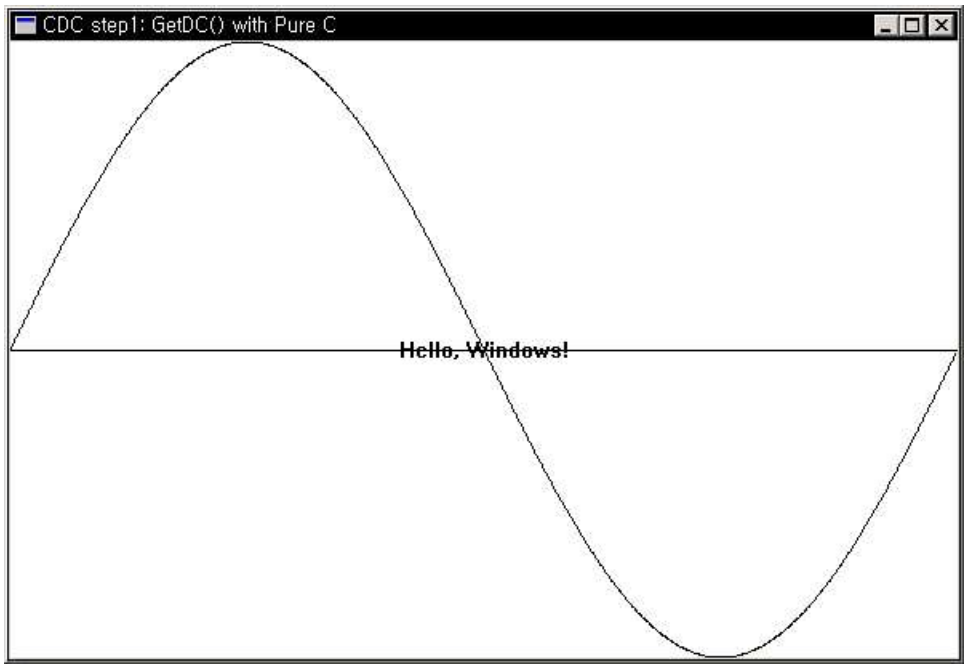
```
LRESULT CView::OnSize(WPARAM wParam,LPARAM lParam)
{
    //#define LOWORD(a) ( (a) & 0x0000ffff )
    //#define HIWORD(a) ( ((a) & 0xffff0000) >> 16)
    cxClient = LOWORD(lParam);
    cyClient = HIWORD(lParam);
    return 0L;
}//CView::OnSize
```

그리고 OnLButtonDown()에서 사인 곡선을 그린다.

```
LRESULT CView::OnLButtonDown(WPARAM wParam,LPARAM lParam)
{
    int i;

    hdc = GetDC(hwnd);

    MoveToEx (hdc, 0,         cyClient / 2, NULL) ;
    LineTo   (hdc, cxClient, cyClient / 2) ;

    for (i = 0 ; i < NUM ; i++)
    {
```

```
        pt[i].x = i * cxClient / NUM ;
        pt[i].y = (int) (cyClient / 2 *
            (1 - sin (TWOPI * i / NUM))) ;
    }//for
    Polyline (hdc, pt, NUM) ;

    ReleaseDC(hwnd,hdc);
    return 0L;
}//CView::OnLButtonDown
```

모든 GDI 함수가 디바이스 컨텍스트 핸들 hdc를 파라미터로 요구하는 것에 주목하라. 실행 결과는 아래 [그림 8.1]과 같다.



[그림 8.1] GetDC()를 이용한 사인 곡선(sine curve)의 출력: 모든 그리기 함수는 HDC를 파라미터로 요구한다.

CView클래스 관련 소스를 [예제 8.1,2]에 리스트하였다. 나머지 소스는 이전의 예와 같다.

[예제 8.1] CView.h

```
#include <windows.h>
#include "stdafx.h"
#include "CObject.h"

#ifndef _CView_
#define _CView_

class CView;

// message map ----------------------------------------------------
typedef LRESULT (CView::*CViewFunPointer)(WPARAM,LPARAM);

typedef struct tagMessageMap
{
    UINT iMsg;
    CViewFunPointer fp;
} MessageMap;

static CViewFunPointer fpCViewGlobal;//pointer to a member function

//{{seojt
#define NUM    1000
#define TWOPI  (2 * 3.14159)
//}}seojt

// class CView ----------------------------------------------------
class CView : public CObject
{
public:
    //{{seojt
    PAINTSTRUCT ps;
    POINT pt[NUM];
    HDC hdc;
    int cyClient;
    int cxClient;
    //}}seojt

    //{{AFX_MESSAGE
    LRESULT OnCreate(WPARAM,LPARAM);
    LRESULT OnDraw(WPARAM,LPARAM);
    LRESULT OnDestroy(WPARAM,LPARAM);
```

```
    //{{seojt
    LRESULT OnSize(WPARAM wParam,LPARAM lParam);
    LRESULT OnLButtonDown(WPARAM wParam,LPARAM lParam);
    //}}seojt
    //}}AFX_MESSAGE

    DECLARE_MESSAGE_MAP()
};//class CView

#endif
```

[예제 8.2] CView.cpp

```
#include <windows.h>
#include <math.h>
#include "stdafx.h"
#include "CView.h"

CView app;

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

LRESULT CView::OnCreate(WPARAM wParam,LPARAM lParam)
{
    return 0L;
}//CView::OnCreate

LRESULT CView::OnDraw(WPARAM wParam,LPARAM lParam)
{
    HDC         hdc;
    PAINTSTRUCT ps;
```

```
    RECT        rect;

    hdc = BeginPaint(hwnd,&ps);
    GetClientRect(hwnd,&rect);
    DrawText(hdc,"Hello, Windows!",-1,&rect,
        DT_SINGLELINE¦DT_CENTER¦DT_VCENTER);
    EndPaint(hwnd,&ps);
    return 0L;
}//CView::OnDraw

LRESULT CView::OnDestroy(WPARAM wParam,LPARAM lParam)
{
    PostQuitMessage(0);
    return 0L;
}//CView::OnDestroy

//{{seojt
LRESULT CView::OnLButtonDown(WPARAM wParam,LPARAM lParam)
{
    int i;

    hdc = GetDC(hwnd);

    MoveToEx (hdc, 0,         cyClient / 2, NULL) ;
    LineTo   (hdc, cxClient, cyClient / 2) ;

    for (i = 0 ; i < NUM ; i++) // (1)
    {
        pt[i].x = i * cxClient / NUM ;
        pt[i].y = (int) (cyClient / 2 *
            (1 - sin (TWOPI * i / NUM))) ;
    }//for
    Polyline (hdc, pt, NUM) ; // (2)

    ReleaseDC(hwnd,hdc);
    return 0L;
}//CView::OnLButtonDown
//}}seojt

//{{seojt
LRESULT CView::OnSize(WPARAM wParam,LPARAM lParam)
{
    //#define LOWORD(a) ( (a) & 0x0000ffff )
```

```
    //#define HIWORD(a) ( ((a) & 0xffff0000) >> 16)
    cxClient = LOWORD(lParam);
    cyClient = HIWORD(lParam);
    return 0L;
}//CView::OnSize
//}}seojt
```

사인 곡선을 그리는 부분은 먼저 포인트 리스트를 배열 pt[]에 채우고(1), PolyLine()을 이용하여 직선의 리스트를 그린다(2).

단계 2: CDC의 설계

<절도비라>

이 절에서는 디바이스 컨텍스트 핸들(HDC, handle to device context)를 감싼(wrap) CDC 클래스를 설계한다. CDC를 설계할 때 객체를 만들고 소멸하는 과정에서 자동으로 GetDC()와 ReleaseDC()가 호출되도록 하여, 그리기 함수의 전후에 DC의 생성과 소멸에 관한 명시적인 호출을 생략하는 방법도 익힌다. 또한 CDC 클래스를 이용하여 단계 1에서 작성한 소스를 개선한다.

</절도비라>

우리는 GDI함수의 래퍼 클래스(wrapper class)를 제작할 수 있다. BeginPaint()와 GetDC()를 호출하여 HDC를 얻을 때 모두 윈도우 핸들을 필요로 한다. 래퍼 클래스의 생성자에서 이 핸들을 얻기 위해 CView 객체의 시작 주소가 필요하다. 그래서 CDC클래스의 생성자를 다음과 같이 작성할 수 있다.

```
CDC::CDC(CView* p)
{
    pView = p;
    hdc   = GetDC(pView->hwnd);
```

<주의>

뷰의 윈도우 핸드을 직접 접근하는 것은 좋은 방법이 아니다. 후에 이러한 사항을 개선할 것이다.

```
    </주의>
}
```

생성자에서는 뷰 객체의 시작 주소를 초기화하고, HDC를 얻기 위해 GetDC()를 호출한다. 파괴자에서는 HDC를 반환하기 위해 ReleaseDC()를 호출한다.

```
CDC::~CDC()
{
    ReleaseDC(pView->hwnd,hdc);
}
```

CDC 클래스의 소스는 [예제 8.3,4]와 같다.

[예제 8.3] DC.h

```
// DC.h: interface for the CDC class.
//
//////////////////////////////////////////////////

#include "CView.h"

#if !defined(_CDC_DEFINED_)
#define _CDC_DEFINED_

class CDC
{
private:
    CView* pView; // (1)
    PAINTSTRUCT ps;
    HDC hdc; // (2)

public:
    CDC(CView* pView);
    virtual ~CDC();
    BOOL MoveToEx(int, int, LPPOINT);
    BOOL LineTo(int, int);
    BOOL Polyline (CONST POINT*, int);
};//class CDC

#endif // !defined(_CDC_DEFINED_)
```

[예제 8.4] DC.cpp

```
// DC.cpp: implementation of the CDC class.
//
//////////////////////////////////////////////////

#include "DC.h"
#include "CView.h"

CDC::CDC(CView* p)
{
    pView = p;
    hdc   = GetDC(pView->hwnd);
}

CDC::~CDC()
{
    ReleaseDC(pView->hwnd,hdc);
}

BOOL CDC::MoveToEx(int x, int y, LPPOINT lpPoint)
{
    return ::MoveToEx(hdc, x, y, lpPoint);
}

BOOL CDC::LineTo(int x, int y)
{
    return ::LineTo(hdc, x, y);
}

BOOL CDC::Polyline (CONST POINT* lppt, int cPoints)
{
    return ::Polyline(hdc, lppt, cPoints);
}
```

구현된 예에서 모든 GDI함수를 랩(wrap)하지는 않았다. 하지만, MFC는 모든 GDI함수를 랩하고 있다. pView는 CDC객체를 생성한 윈도우의 핸들을 얻기 위해 필요하다(1). 멤버 hdc는 실제 GDI함수를 호출할 때 전달된다(2).

이제 사인 곡선을 그리는 부분은 CDC객체를 만들고 CDC 객체의 멤버를 호출함으로써 구현한다.

```
LRESULT CView::OnLButtonDown(WPARAM wParam,LPARAM lParam)
{
    CDC dc(this); // (1)
    int i;

    dc.MoveToEx (0, cyClient / 2, NULL) ; // (2)
    dc.LineTo(cxClient, cyClient / 2) ;

    for (i = 0 ; i < NUM ; i++)
    {
        pt[i].x = i * cxClient / NUM ;
        pt[i].y = (int) (cyClient / 2 *
            (1 - sin (TWOPI * i / NUM))) ;
    }//for
    dc.Polyline (pt, NUM) ; // (2)

    return 0L; // (3)
}//CView::OnLButtonDown
```

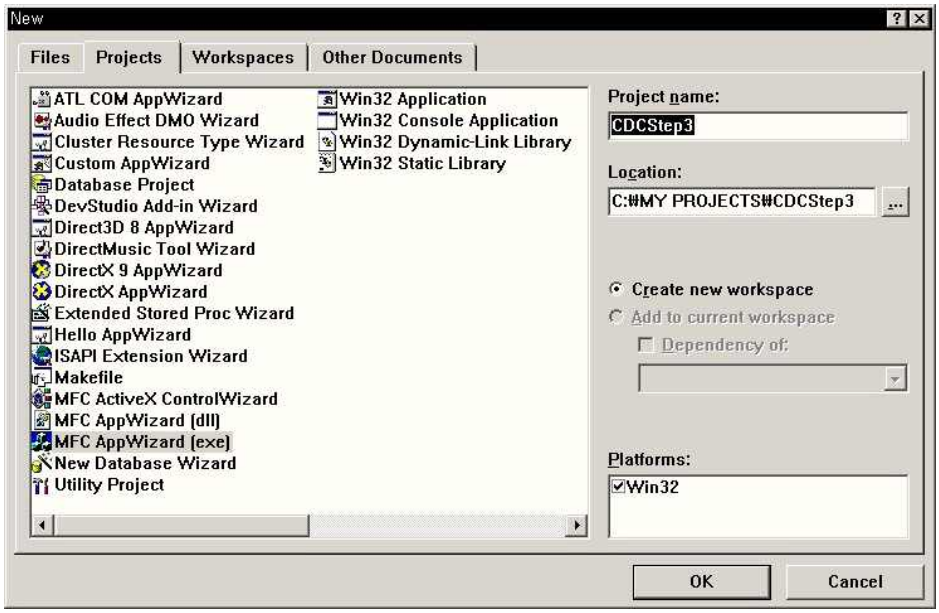
CDC 객체를 만들 때는 윈도우 핸들을 얻기 위해 CView의 시작 주소를 필요로 한다. CView의 멤버에서 이것은 this이다(1). 래핑(wrapping)된 모든 멤버 함수에서 HDC는 숨겨진다(2). 함수를 빠져나가기 전에 ReleaseDC()를 호출할 필요가 없다. 이것은 dc객체가 파괴될 때 CDC의 파괴자에서 호출□된다(3).

<여기서 잠깐>
CDC 클래스의 예에서처럼 생성자와 파괴자의 특별한 동작을 이용하면 복잡한 코드를 간단하게 표현할 수 있다. 대표적인 예로 동기화 객체인 CRITICAL_SECTION을 감싸는 코드는 생성자와 파괴자에서 각각 EnterCriticalSection()과 LeaveCriticalSection()을 호출하므로, 프로그래머의 실수를 방지한다. 또한, 복잡한 컨트롤에 의해서 다양한 탈출 경로를 가지는 함수를 빠져나가기 전에 실행해야할 코드가 있다면, 콜백을 등록하고 파괴자에서 호출되도록 함으로써 코드의 유지/보수를 쉽게 할 수 있다.
</여기서 잠깐>

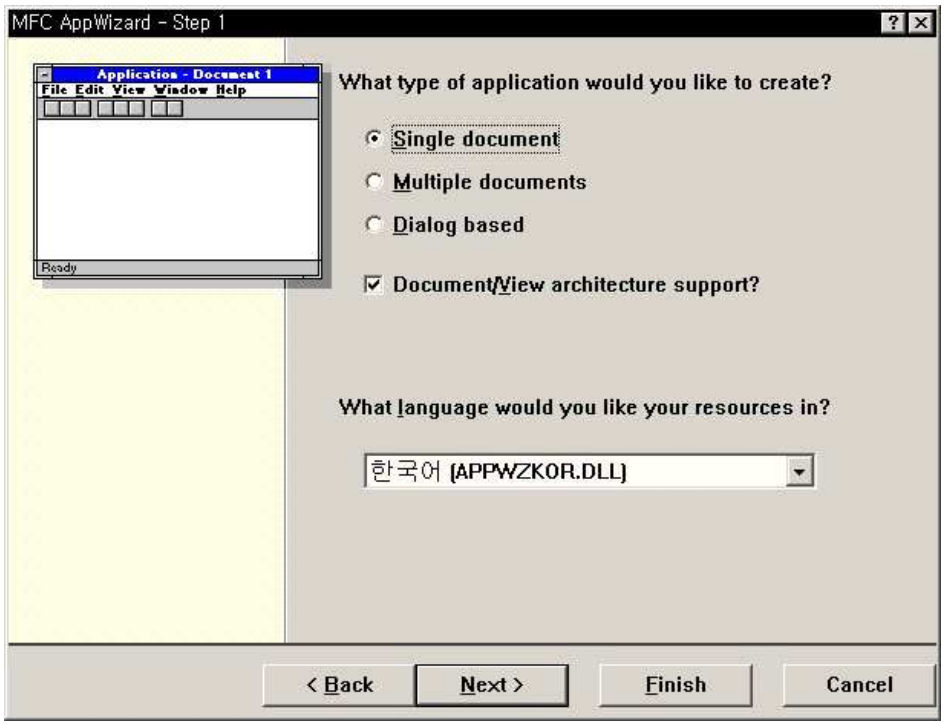
MFC 코드의 작성

<절도비라>
이 절에서는 단계 2에서 작성한 사인 곡선을 그리는 프로그램을, MFC를 이용하여 작성해 보면서, 단계 2와 크게 다르지 않음을 확인한다. 또한, MFC의 CClientDC의 계층 구조를 이해하여, DC의 공통 부분이 CDC에 구현되어 있음을 확인한다.
</절도비라>

이제 같은 버전을 MFC로 만들어 보자. 이름을 CDCStep3으로 하여, MFC 싱글 다큐먼트(single document) 프로젝트를 만든다.



[그림 8.2] CDCStep3 프로젝트 만들기: MFC AppWizard(exe) 프로젝트를 만든다. 프로젝트 이름을 CDCStep3으로 입력한 다음, OK 버튼을 선택한다.



[그림 8.3] Single document 선택: Single Document 라디오 버튼을 선택하고 Finish버튼을 선택하여, AppWizard가 디폴트 코드를 생성하도록 한다.

이제 Ctrl+W를 눌러 클래스 위저드(class wizard)를 실행한다. 그리고 WM_LBUTTONDOWN과 WM_SIZE메시지를 뷰 클래스 CDCStep3View에 맵하여 코드를 생성한다□.

<여기서 잠깐>
응용 프로그램의 골격을 생성하는 코드 자동 생성기를 AppWizard, 생성된 코드에서 멤버 함수를 유지/보수하는 코드 자동 생성기를 클래스 위저드라 한다.
</여기서 잠깐>

그러면 뷰 클래스 헤더 파일의 다음과 같은 코드는

```
...
protected:

// Generated message map functions
```

```
protected:
    //{{AFX_MSG(CCDCStep3View)
        // NOTE - 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()
};
...
```

아래와 같이 굵게 표시한 부분의 코드가 추가된다.

```
...
protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CCDCStep3View)
    afx_msg void OnSize(UINT nType, int cx, int cy);
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

그리고 뷰 클래스의 구현 파일(CDCStep3.cpp)에도 코드가 생성된다. 메시지 맵에 두개의 핸들러에 대한 엔트리가 추가된다.

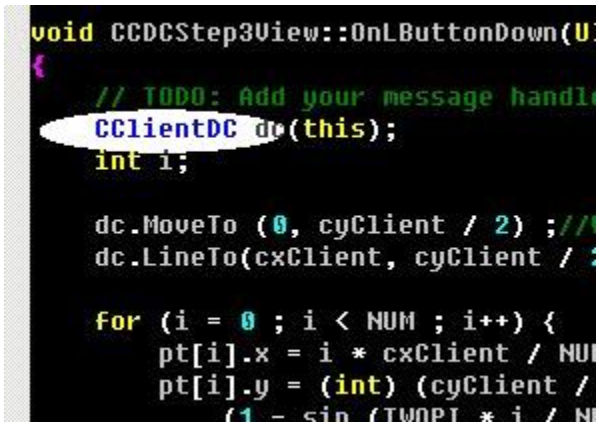
```
BEGIN_MESSAGE_MAP(CCDCStep3View, CView)
    //{{AFX_MSG_MAP(CCDCStep3View)
    ON_WM_SIZE()
    ON_WM_LBUTTONDOWN()
    //}}AFX_MSG_MAP
    // Standard printing commands
    ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
    ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()
```

이제 멤버 함수의 몸체를 작성한다.

```
void CCDCStep3View::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    // TODO: Add your message handler code here
    cxClient=cx;
    cyClient=cy;
    //}}seojt

}

void CCDCStep3View::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    CClientDC dc(this);
    int i;

    dc.MoveTo (0, cyClient / 2) ;//Why MoveToEx() isn't exist?
    dc.LineTo(cxClient, cyClient / 2) ;

    for (i = 0 ; i < NUM ; i++)
    {
        pt[i].x = i * cxClient / NUM ;
        pt[i].y = (int) (cyClient / 2 *
            (1 - sin (TWOPI * i / NUM))) ;
    }//for
    dc.Polyline (pt, NUM) ;
    //}}seojt

    CView::OnLButtonDown(nFlags, point);
}
```

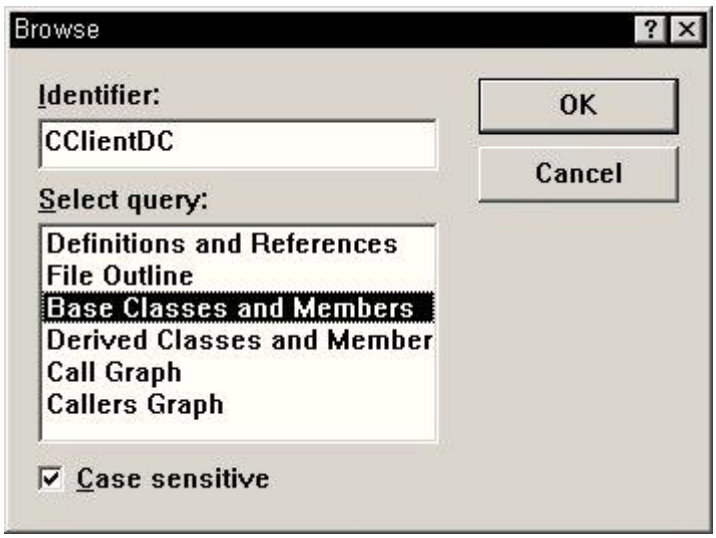
DC객체의 이름이 **CClientDC**인 것에 주목하라. HDC(handle to device context)를 얻는 각각의 방법에 대해 디바이스 컨텍스트 객체가 존재한다. 그 중 CClientDC는 GetDC()로 얻는 디바이스 컨텍스트를 위한 객체이다. 위의 예에서는 핸들러가 처리하는 메시지가 WM_LBUTTONDOWN이므로 CClientDC 타입의 객체를 만들어야 한다.

CClientDC의 계층 구조를 따라가 보도록 하자. CClientDC위에서 Alt+F12를 누른다.



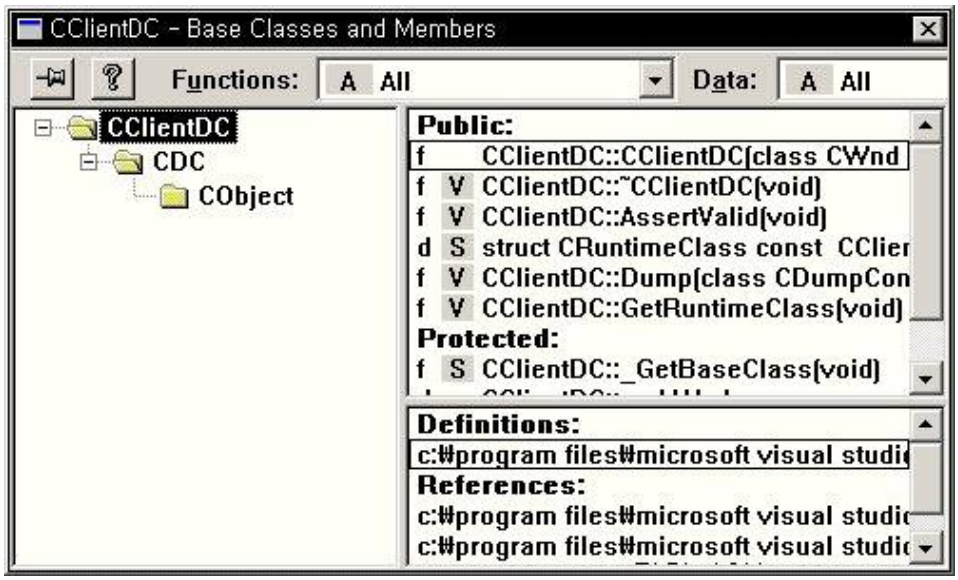
[그림 8.4] CClientDC의 브라우징: 클래스 이름 위에서 Alt+F12를 누르면 소스 브라우즈 선택창이 나타난다.

[그림 8.5]처럼 나타나는 브라우즈 창에서 Base Classes and Members를 선택한다. 그러면 CClientDC의 베이스 클래스를 확인할 수 있다.



[그림 8.5] CClientDC의 베이스 클래스 확인:

그러면 CClientDC의 베이스 클래스가 CDC인 것을 알 수 있다. 이제 CDC를 선택해 보자. 그러면 MFC 소스에서 CDC가 정의된 소스로 이동한다.



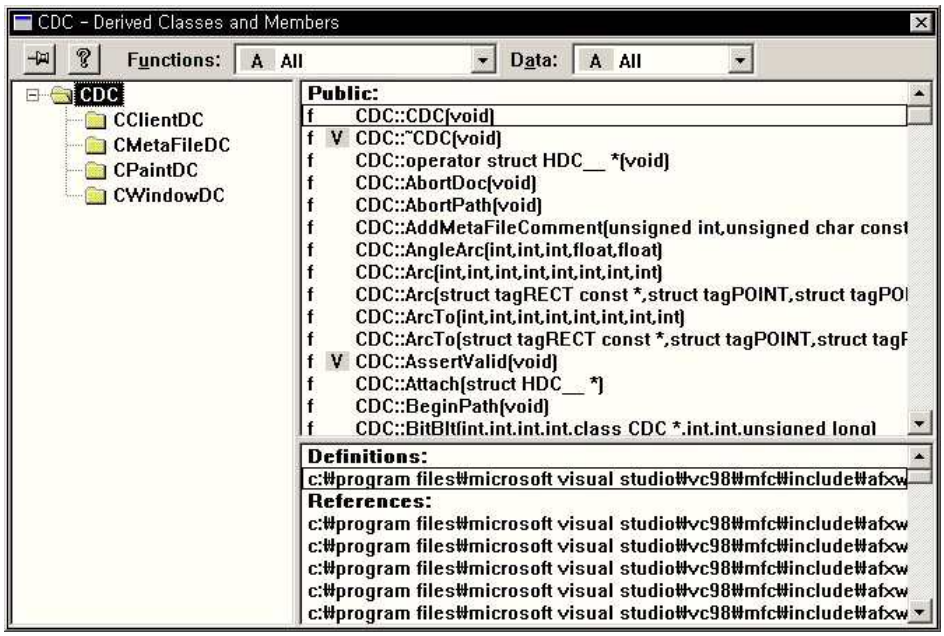
[그림 8.6] CClientDC의 계층 구조:

MFC 소스의 CDC에서 다시 Alt+F12를 누르면 나타나는 브라우즈 창에서 이번에는 Derived Classes and Members를 선택한다.

[그림 8.7] CDC의 소스 확인:

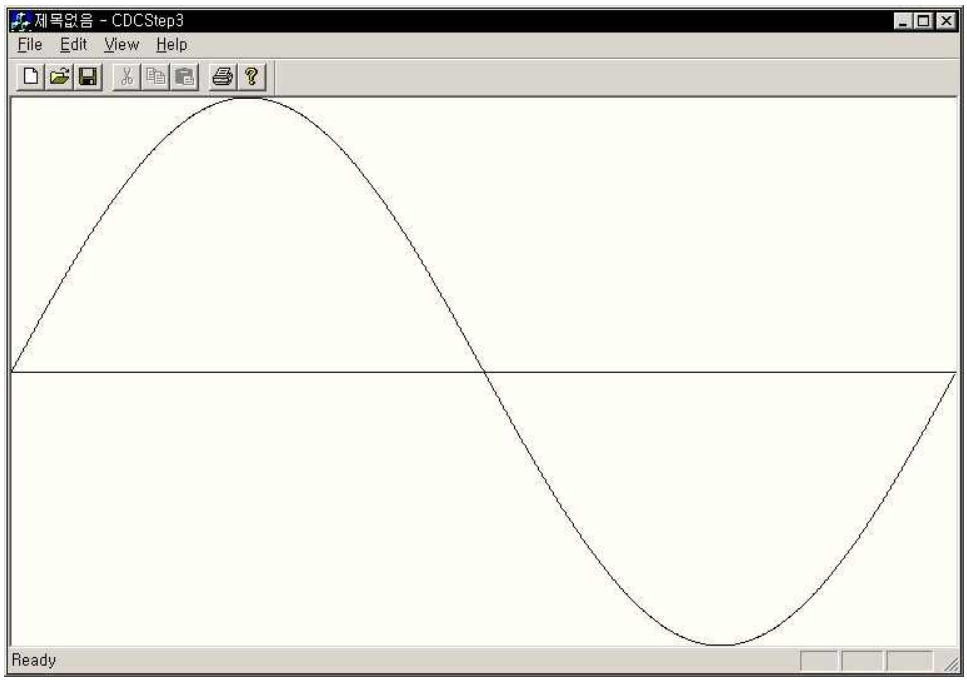
그러면 CDC를 상속받아 구현한 클래스에는 네 가지가 있는 것을 확인할 수 있다. 그 중 CClientDC와 CPaintDC에 주목하자. 윈도우 메시지가

WM_PAINT일 때에는 CPaintDC로 객체를 만들어야 하고, WM_PAINT이외의 모든 메시지에 대해서는 CClientDC로 DC객체를 만들어야 한다는 것은 1장에서 설명한 바 있다.



[그림 8.8] CDC의 하위 클래스들: CClientDC는 생성자에서 GetDC()를 호출한다. CPaintDC는 생성자에서 BeginPaint()를 호출한다.

OnLButtonDown()에서 DC객체를 만들지 않고, GetDC()를 직접 호출해서 그리기를 시도해도 좋다. 독자들이 직접 꼭 해보기 바란다. 실행 결과는 아래 [그림 8.9]와 같다.



[그림 8.9] 사인 곡선의 MFC버전의 실행 화면: 마우스 왼쪽 버튼을 누르면 클라이언트 영역에 사인 곡선을 그린다.

MFC 코드의 확인

<절도비라>
이제까지 우리는 GDI의 래퍼 클래스인 CClientDC의 원리를 살펴보고 간단하게 구현해 보았다. 이제 MFC 소스에서 직접 이 부분의 코드를 확인해 보자.
</절도비라>

이제 MFC 소스를 확인해 볼 차례이다. AFXWIN.H의 1035번 줄에 CClientDC 클래스의 선언을 확인할 수 있다. CDC를 상속받는 간단한 클래스로 윈도우 핸들을 멤버로 가지고 있다.

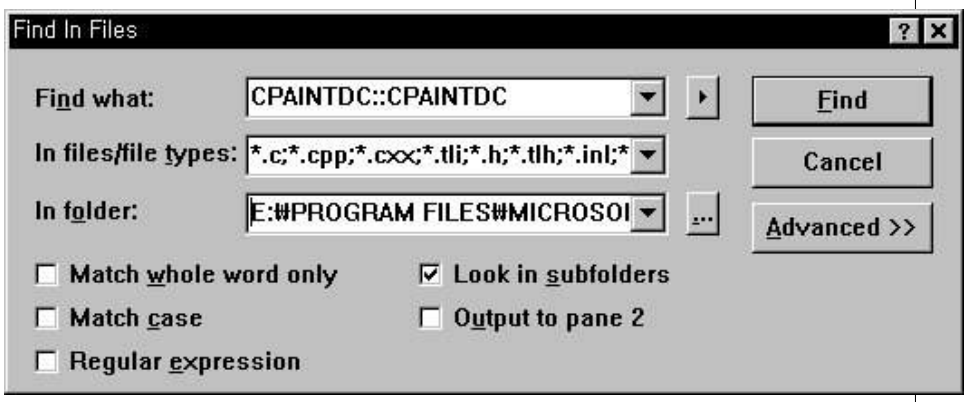
```
class CClientDC : public CDC
{
```

```
    DECLARE_DYNAMIC(CClientDC)

// Constructors
public:
    CClientDC(CWnd* pWnd);

// Attributes
protected:
    HWND m_hWnd;

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

생성자와 파괴자 코드를 확인해 보면(WINGDI.CPP의 979번 줄)□, 예상대로

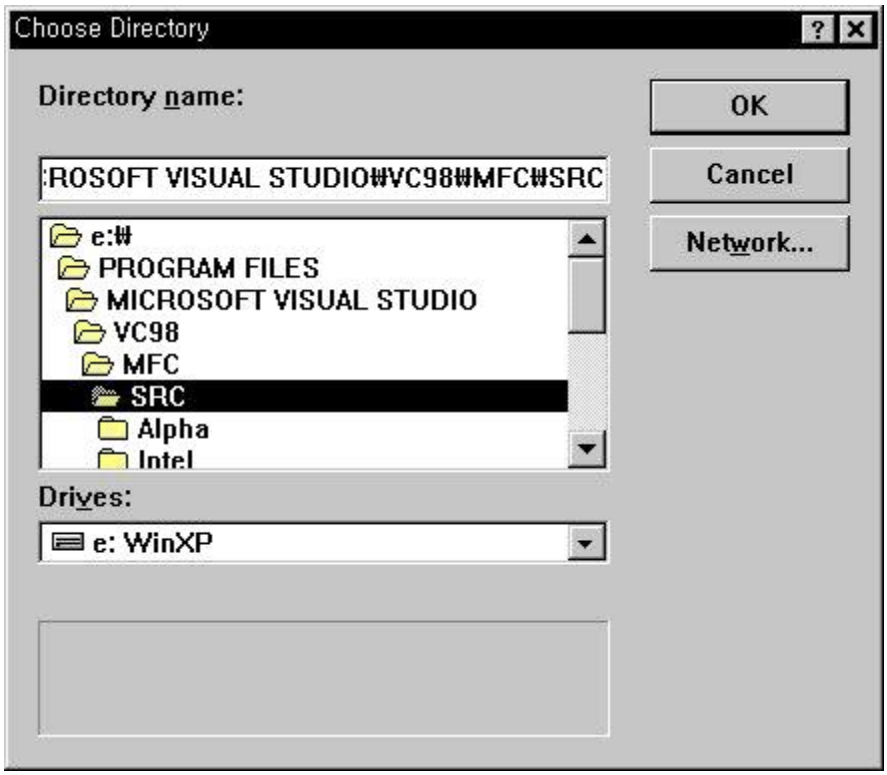
<여기서 잠깐>

MFC 소스에서 문자열을 찾기 위해 Find In Files... 기능을 이용한다. Edit→Find In Files...를 선택하면 파일에서 찾기 대화상자를 실행한다.



[그림 8.10] Find In Files 대화상자

Find what: 텍스트 상자에 찾고자 하는 문자열을 입력하고, In folder:에서 ... 버튼을 선택하여 대상 폴더를 아래와 같이 지정한다.



[그림 8.11] Choose Directory 대화상자

이제 MFC의 소스에서 문자열을 찾을 수 있다. 찾은 결과를 차례대로 확인하는 키는 F4이다.
</여기서 잠깐>

GetDC()와 ReleaseDC()를 호출하고 있음을 알 수 있다. 윈도우 핸들을 그대로 사용하지 않고, GetSafeHwnd()로 항상 m_hWnd의 값을 갱신함에 주목하라. 이 이유를 후에 RTTI를 구현하면서 살펴 볼 것이다.

```
CClientDC::CClientDC(CWnd* pWnd)
{
    ASSERT(pWnd == NULL || ::IsWindow(pWnd->m_hWnd));

    if (!Attach(::GetDC(m_hWnd = pWnd->GetSafeHwnd())))
        AfxThrowResourceException();
}
```

```
CClientDC::~CClientDC()
{
    ASSERT(m_hDC != NULL);
    ::ReleaseDC(m_hWnd, Detach());
}
```

이제 CPaintDC 클래스의 생성자와 파괴자를 확인해 보자. WINGDI.CPP의 1043번 줄에 정의된 생성자와 파괴자를 보면, 각각 BeginPaint()와 EndPaint()를 호출하고 있음을 확인할 수 있다.

```
CPaintDC::CPaintDC(CWnd* pWnd)
{
    ASSERT_VALID(pWnd);
    ASSERT(::IsWindow(pWnd->m_hWnd));

    if (!Attach(::BeginPaint(m_hWnd = pWnd->m_hWnd, &m_ps)))
    AfxThrowResourceException();
}

CPaintDC::~CPaintDC()
{
    ASSERT(m_hDC != NULL);
    ASSERT(::IsWindow(m_hWnd));

    ::EndPaint(m_hWnd, &m_ps);
    Detach();
}
```

이제 모든 GDI함수가 래핑(wrapping)된 CDC 클래스의 소스를 볼 순서이다. 클래스의 선언을 AFXWIN.H의 636번 줄에서 1007번 줄까지 확인할 수 있다. 소스는 아래와 같다. 긴 소스를 모두 리스트하지는 않았으므로 독자들이 Find In Files... 기능으로 꼭 직접 확인해 보기 바란다.

```
class CDC : public CObject
{
    DECLARE_DYNCREATE(CDC)
public:

// Attributes
    HDC m_hDC; // The output DC (must be first data member)
```

```
// 이하 생략
// ...
};
```

요약

MFC에서 그리기는 CClientDC 객체를 만들어 진행한다. 메시지가 WM_PAINT인 경우는 CPaintDC를 만들어야 하지만, CPaintDC 객체를 만들일은 거의 없다. 왜냐하면, WM_PAINT 메시지 핸들러 **OnPaint()**는 이미 MFC의 CView 클래스에 구현되어 있고, 이 핸들러가 가상함수 OnDraw()를 호출하는데, OnDraw()의 파라미터로 이미 DC 객체가 전달되기 때문이다.

CView의 OnPaint()의 소스는 아래와 같다.

```
void CView::OnPaint()
{
    // standard paint routine
    CPaintDC dc(this);
    OnPrepareDC(&dc);
    OnDraw(&dc);
}
```

그러므로 뷰에 그리는 작업의 대부분을 OnDraw()에서 처리한다. 하지만 더블 버피링(double buffering) 등의 특별한 처리를 위해서는 OnPaint()를 직접 코딩할 일이 방생하므로, OnPaint()에서는 CPaintDC 객체를 만들어야 한다는 사실을 잊지 말자.

모든 GDI함수는 DC객체에 랩되어 있으므로 DC객체의 멤버를 사용하면 되지만, 윈도우즈의 API함수를 바로 사용해도 좋다.

- **CClientDC**등의 DC 클래스는 GDI 함수들의 래퍼 클래스이다. CDC는 모든 DC 클래스의 공통 조상이다.
- **DC 클래스의 객체를 생성**하기 위해서 생성자에 전달해야 하는 파라미터는 CWnd의 하위 클래스 객체의 시작주소이며, 대부분의 문맥에서 이것은 this이다.

- OnPaint()의 내부에서 OnDraw()를 호출한다. OnDraw() 이외의 곳에서 그리기를 하기 위해서 **CPaintDC** 타입의 객체를 만들어서는 안 된다. 왜냐하면 CPaintDC는 WM_PAINT를 위한 전용 DC 클래스이기 때문이다.

[[[[["비주얼 C++ 애드인"은 절이 아닌 특별한 구성 요소('숙련된 개발자가 되기 위한 필수 애드인')로 처리 요망]]]]]

비주얼 C++ 애드인(Add-ins)

<절도비라>

이 절에서는 비주얼 C++에서 많이 사용되는 애드인 프로그램을 살펴본다. 훌륭한 개발자는 프로그래밍 실력과 논리력이 뛰어나야 하는 것은 물론이고 좋은 개발 툴들을 능숙하게 사용할 수 있어야 할 것이다.

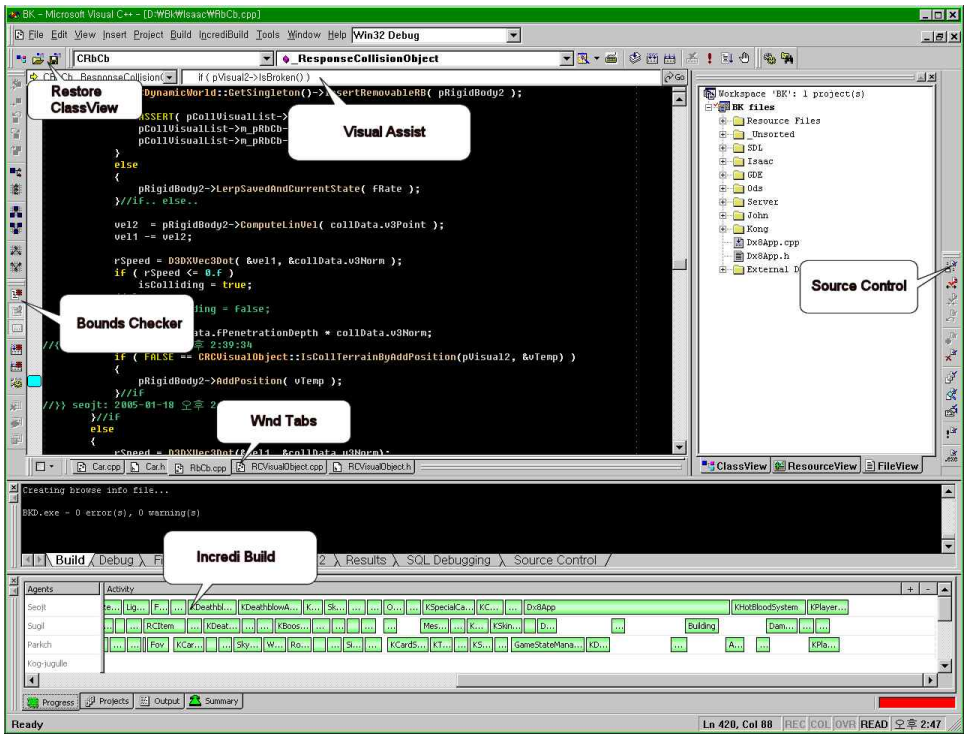
</절도비라>

필자가 비주얼 C++과 함께 사용하는 **애드인(add-in)** 프로그램□을 소개한다. 애드인은 비주얼 C++의 기능을 확장하기 위한 DLL 프로그램이다.

<저자한마디>
현업에서 응용 프로그램 - 게임 - 을 개발하면서, 툴의 중요성을 실감한다. 훌륭한 프로그래머는 잘 정의된 설계를 작성하고, 문제 발생시 해결 능력이 뛰어나기도 해야겠지만, 좋은 툴들을 잘 사용할 줄 알아야 할 것 같다. 한 예를 들면, 코드 프로젝트(www.codeproject.com)에서 알게 된, 크래시리포트(CrashReport)라는 툴을 사용함으로써, 프로그램이 죽은 후에도 디버깅이 가능하게 되었다! 디버깅을 언제 하냐고? 프로그램이 죽으면 그 때하면 되는 것이다!
</저자한마디>

가장 대중적인 애드인으로 **바운즈 체커(Bounds Checker)**와 **비주얼 어시스트(Visual Assist)**가 있다. 바운즈 체커를 이용하면 메모리 릭과 메모리 오버런(memory overrun)과 관련된 대부분의 에러를 자동으로 잡아낼 수 있다. 비주얼 어시스트는 강력한 편집 보조 기능을 제공한다. 멤버 함수를 자동으로 찾아가는 기능이나, 멤버에 대한 문법 강조, 향상된 자동 완성기능은 너무나 훌륭한 기능이다.

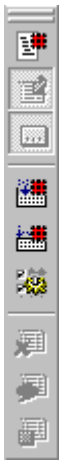
아래 그림은 필자의 비주얼 C++ IDE 화면이다(사실 2개의 모니터를 쓰고 있다. 한쪽에는 코드 위주로 정렬된 화면, 다른 한쪽에는 디버깅등 보조기능용이다. 하지만, 설명을 위해, 한 화면에 모든 애드인이 나타나도록 하였다).



[그림 8.12] 비주얼 C++ 통합환경: 가장 요긴한 툴은 바운즈체커, 인크레디 빌드(IncrediBuild), 퍼포스(Perforce), 비주얼 어시스트이다. 독자들의 IDE에 포함된 애드인은 Tools->Customize...를 선택한 다음, Add-ins... 탭을 선택하면 확인해 볼 수 있다(몇 가지 애드인은 www.codeproject.com에서 구할 수 있다).

바운즈 체커(Bounds Checker)

바운즈 체커는 메모리 릭(memory leak), 메모리 오버런(memory overrun) 등의 에러를 자동으로 발견하는 요긴한 툴이다. 프로젝트의 선임자가 작성해 놓은 코드에서 어디에서 발생하는지 모르는 메모리 릭 때문에 밤새워본 사람들이라면, 바운즈 체커의 위력을 실감할 수 있을 것이다. 3일간의 밤샘을 바운즈 체커는 단 5분 만에 해결할 수 있다!

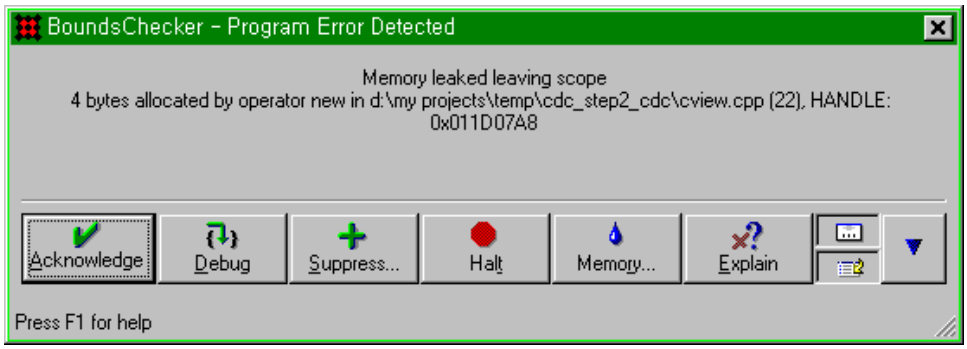


[그림 8.13] 바운즈 체커의 툴바 화면: 바운즈 체커는 메모리 릭뿐만 아니라, 다양한 종류의 에러와 에러 가능성을 발견하는 유용한 툴이다.

부록 CD-ROM에 제공되는 8장의 /CDC_step2_CDC 폴더를 열어서 바운즈 체커가 어떻게 동작하는지 확인해 보자. CDCSTEP2.DSW 프로젝트를 연다. 그리고 CView.cpp 파일의 OnCreate()를 다음과 같이 수정한다.

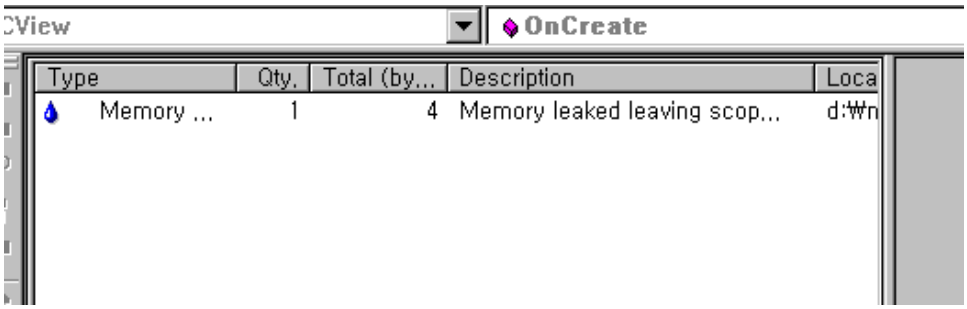
```
LRESULT CView::OnCreate(WPARAM wParam,LPARAM lParam) {
    int* p = new int(); // 동적 메모리 할당

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

소스를 보면 함수 내부에서 힙에 메모리 할당을 한 뒤, 할당한 메모리를 클리어 하지 않고, 함수를 탈출하는 논리 에러가 포함된 것을 확인할 수 있다. 이제 바운즈 체커 툴바의 첫번째 버튼을 눌러 바운즈 체커를 활성화 시키고, F5를 눌러 디버그 런 한다. 그러면 바운즈 체커가 아래와 같은 에러 윈도우를 표시한다.



[그림 8.14] 바운즈 체커의 메모리 릭 발견: 바운즈 체커는 함수 스코프를 빠져 나가는 순간 메모리 릭이 발생했음을 사용자에게 보고한다.

바운즈 체커가 보고한 대화상자에서 Debug버튼을 누르면 메모리 릭이 발생한 시점으로 이동한다. 또한, 바운즈 체커의 결과창에서 상세한 정보를 얻을 수 있다.



[그림 8.15] 바운즈 체커의 결과창: 소스에서 4바이트의 메모리 릭이 발생했음을 보고한다.

코드위즈(CodeWiz)

코드위즈는 유용한 단축키를 제공하고, 코드 생성을 돕는다. 코드위즈를 설치한 후, Ctrl+1을 누르면, 클래스 정의 파일(*.h)과 구현 파일(*.cpp)을 토글하여 연다. 긴 함수의 중간에서 다음 함수의 시작 부분으로 가기 위해 Ctrl+PgDn을 누르면, 다음 함수의 시작 부분으로 이동한다. 멤버 함수를 선언한 다음, 함수의 몸체를 구현 파일에 자동으로 생성하기 위해 Ctrl+6을 누른

다.

[그림 8.16] 코드위즈의 애드인 메뉴: 각종 편리한 단축키를 제공한다.

코드위즈의 멤버 함수 생성기능을 알아보자. CView.h의 끝 부분에 다음과 같이 함수 프로토타입을 추가한다.

```
    ...
    LRESULT OnLButtonDown(WPARAM wParam,LPARAM lParam);
    //}}seojt
    //}}AFX_MESSAGE

    int CodeWizFunction(int iParam, float fParam);

    DECLARE_MESSAGE_MAP()
};//class CView
```

이제 커서를 CodeWizFunction()이 선언된 줄의 아무 곳에나 위치시키고,

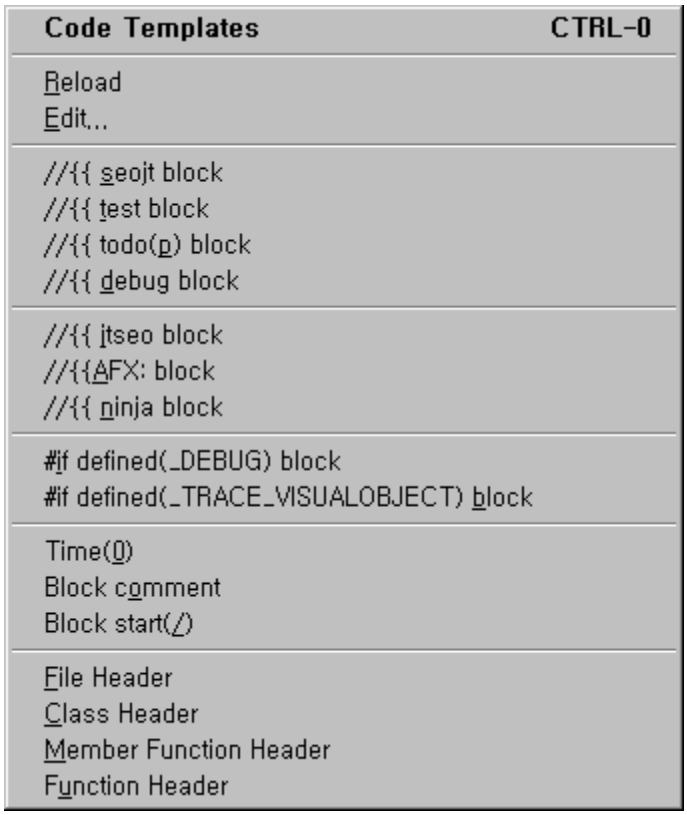
Ctrl+5를 누른다. 그러면 줄이 반전되면서, 멤버 함수에 대한 정보를 코드위즈가 기록해 둔다. 다음에 Ctrl+1을 눌러 CView의 구현 파일을 연다. 그러면 CView.cpp파일이 열린다. 커서를 함수를 생성할 적당한 곳에 위치시키고 Ctrl+6을 누르면 아래와 같이 함수의 몸체가 자동으로 생성된다.

```
int CView::CodeWizFunction(int iParam,float fParam)
{
	//
}
```

코드위즈의 코드 템플릿 기능 또한 유용하다. 코드를 후에 참조하기 위해, 코드의 전후에 //{{ seojt와 //}} seojt를 삽입하기로 했다고 하자. 그러면 코드위즈의 코드 템플릿을 편집해서 해당 기능을 메뉴에 등록한다. 예로 방금 생성한 함수의 몸체의 전후에 주석을 삽입하는 과정을 처리해 보자.

먼저,함수의 몸체를 모두 블록 선택한다. 그리고 코드위즈의 메뉴를 불러 등록된 기능을 활성화 한다(필자의 경우 Ctrl+0,s이다). 그러면 선택한 영역의 앞뒤로 자동으로 코드가 삽입된다. 아래의 코드에서 두 줄의 라인코멘트는 자동으로 삽입된 것이다.

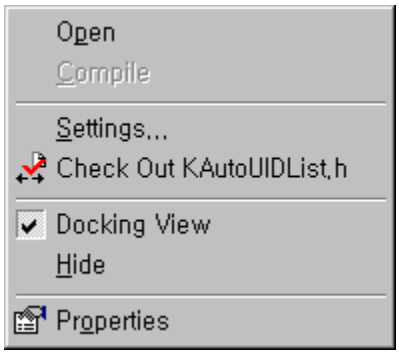
```
//{{ seojt: 2005-01-22 오후 4:47:28
int CView::CodeWizFunction(int iParam,float fParam)
{
    return 0;
}
//}} seojt: 2005-01-22 오후 4:47:28
```



[그림 8.17] 코드위즈의 코드템플릿 메뉴: 자주 사용하는 코드 템플릿을 등록해 사용하면, 번거로운 타이핑을 대폭 줄일 수 있다.

퍼포스(Perforce)

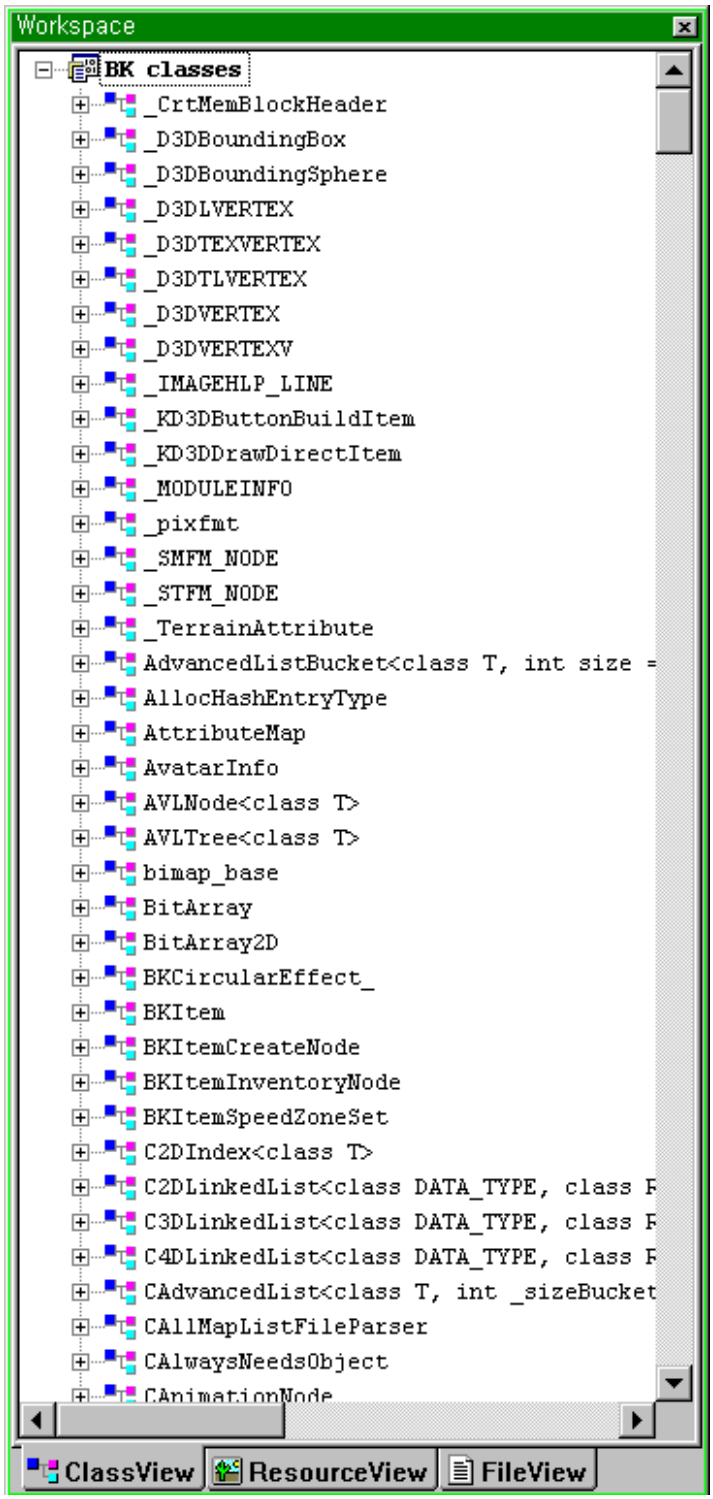
퍼포스는 공동작업에 필요한 **소스 관리 도구(소스 컨트롤, source control)**이다. 한 프로젝트에 필요한 소스 코드를 여러 명이 공유하는 경우, 소스 컨트롤의 사용은 필수적이다. 많은 소스 컨트롤이 있지만, 퍼포스의 장점은 강력한 소스 머지(source merge) 기능이다. 같은 소스를 2명이서 편집한 경우 발생한 소스의 충돌을 퍼포스는 훌륭하게 자동 머지(auto merge)한다(물론 자동으로 머지 되지 않는 경우, 머지 툴이 실행되어 수동을 머지해 주어야 한다).



[그림 8.18] 퍼포스 메뉴: 등록된 소스의 편집을 시작하기 위해서 소스 저장소에서 가져오는 것을 체크 아웃(check out)이라 한다. 체크 아웃된 소스의 편집이 끝나면 체크 인(check in)해 주면, 다른 사람은 가장 최근의 소스를 얻을 수 있다. 퍼포스는 여러 명이 동시에 같은 파일을 체크 아웃하는 것을 허용하며, 이러한 파일의 체크인시에 자동 머지를 지원한다.

리스토어 클래스뷰 애드인

잘 정돈된 클래스 뷰의 폴더 구조가 깨어져서 고생한 독자들에게 리스토어 클래스뷰(Restore ClassView)는 필수적이다. 아래의 화면은 필자가 참여하고 있는 범퍼킹(BumperKing) 프로젝트의 깨어진 클래스 뷰 화면이다.

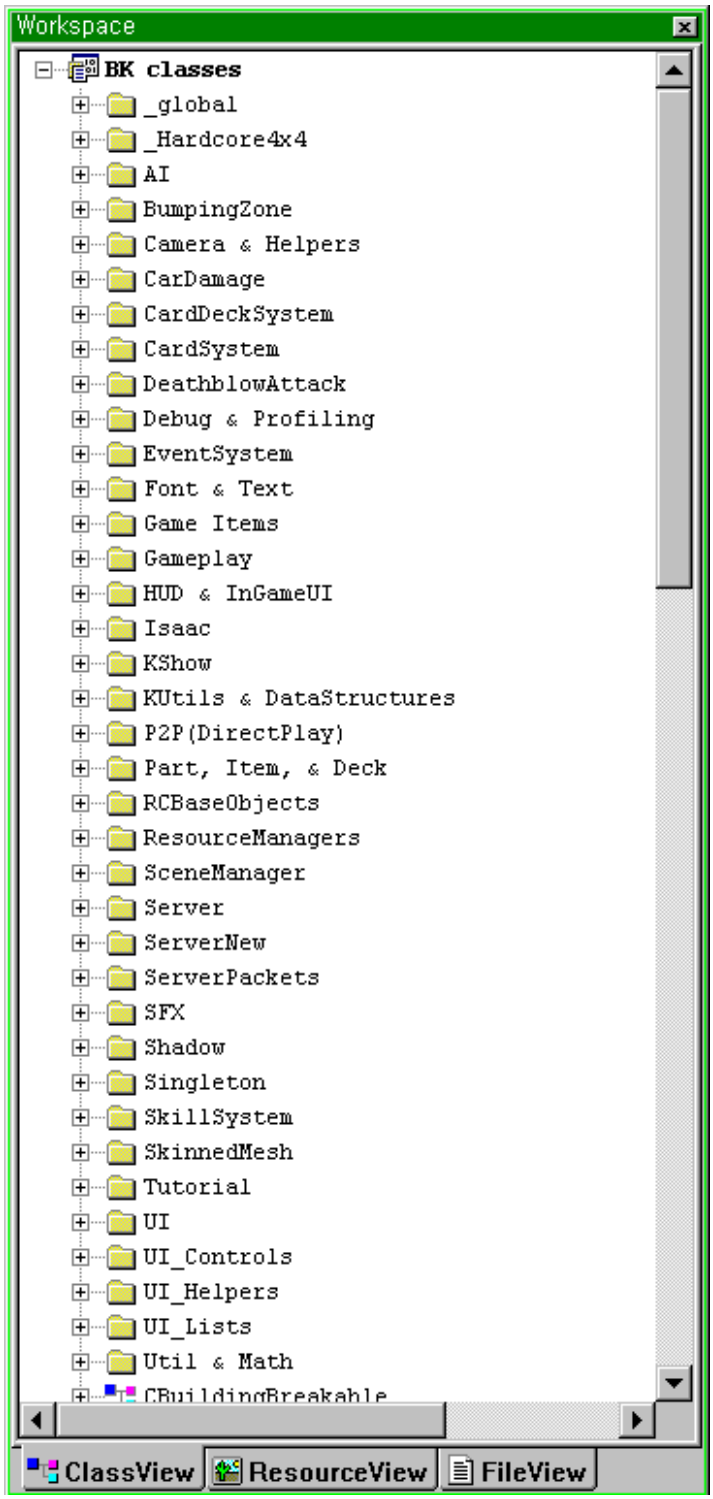

Workspace
BK classes
_CrtMemBlockHeader
_D3DBoundingBox
_D3DBoundingSphere
_D3DLVERTEX
_D3DTEXVERTEX
_D3DTLVERTEX
_D3DVERTEX
_D3DVERTEXV
_IMAGEHLP_LINE
_KD3DButtonBuildItem
_KD3DDrawDirectItem
_MODULEINFO
_pixfmt
_SMFM_NODE
_STFM_NODE
_TerrainAttribute
AdvancedListBucket<class T, int size =
AllocHashEntryType
AttributeMap
AvatarInfo
AVLNode<class T>
AVLTree<class T>
bimap_base
BitArray
BitArray2D
BKCircularEffect_
BKItem
BKItemCreateNode
BKItemInventoryNode
BKItemSpeedZoneSet
C2DIndex<class T>
C2DLinkedList<class DATA_TYPE, class P
C3DLinkedList<class DATA_TYPE, class P
C4DLinkedList<class DATA_TYPE, class P
CAdvancedList<class T, int _sizeBucket
CAllMapListFileParser
CAlwaysNeedsObject
CAnimationNode
ClassView
ResourceView
FileView

[그림 8.19] 깨어진 클래스 뷰 폴더 구조: 리스토어 클래스뷰를 이용하면 깨어진 클래스뷰를 복구할 수 있다.

클래스뷰 폴더 구조를 복구하기 위해, 리스토어 클래스뷰 툴바의 열기 버튼을 선택해서 가장 최근에 저장한 클래스뷰 구조 파일을 불러온다.

[그림 8.20] RestoreClassView: 클래스뷰 폴더 구조의 열기와 저장을 지원한다.

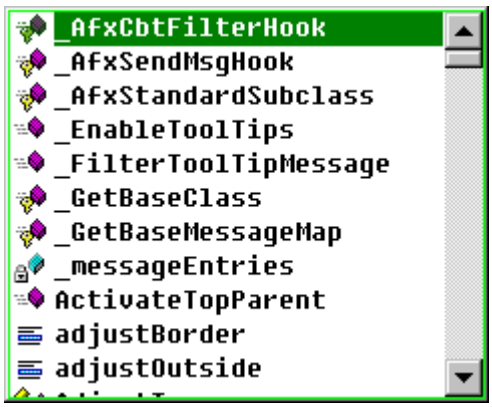
툴바에서 리스토어 클래스뷰 열기 버튼을 선택한 다음, 확장자가 .clv인 가장 최근에 저장해 놓은 파일을 선택하면 클래스뷰 폴더 구조가 복구된다.


Workspace
BK classes
_global
_Hardcore4x4
AI
BumpingZone
Camera & Helpers
CarDamage
CardDeckSystem
CardSystem
DeathblowAttack
Debug & Profiling
EventSystem
Font & Text
Game Items
Gameplay
HUD & InGameUI
Isaac
KShow
KUtils & DataStructures
P2P(DirectPlay)
Part, Item, & Deck
RCBaseObjects
ResourceManagers
SceneManager
Server
ServerNew
ServerPackets
SFX
Shadow
Singleton
SkillSystem
SkinnedMesh
Tutorial
UI
UI_Controls
UI_Helpers
UI_Lists
Util & Math
ClassView
ResourceView
FileView

[그림 8.21] 복구된 범퍼킹 프로젝트의 클래스 뷰: 소스가 1000개를 넘어가는 큰 프로젝트에서 클래스뷰 구조를 복구하는 기능은 많은 도움이 된다.

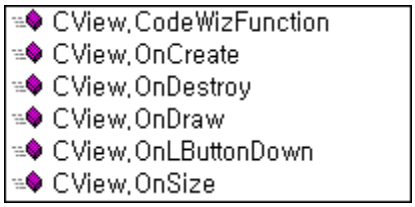
비주얼 어시스트(Visual Assist)

비주얼 어시스트는 소스 작성을 돕는 편리한 툴이다. 대표적인 기능으로 멤버 함수 자동완성 기능, 멤버 함수 찾아가기 기능, 편집 문맥 표시 기능, 프로젝트 파일 열기 기능, 다중 클립보드 기능 등이 있다. 또한, 클래스의 이름과 멤버 함수를 다른 색으로 표시하는 강화된 컬러기능은 한 눈에 함수의 종류를 판단하는데 도움을 준다. 아래의 그림은 비주얼 어시스트의 멤버 함수 자동완성 기능을 보여준다.



[그림 8.22] 비주얼 어시스트의 소스 자동 완성 기능: 객체 이름 다음에 .이나 ->를 적는 순간 팝업되며, 두문자를 타이핑치고, 탭 키를 누르면 여러번의 타이핑을 줄일 수 있고, 타이핑 오류를 미연에 방지한다.

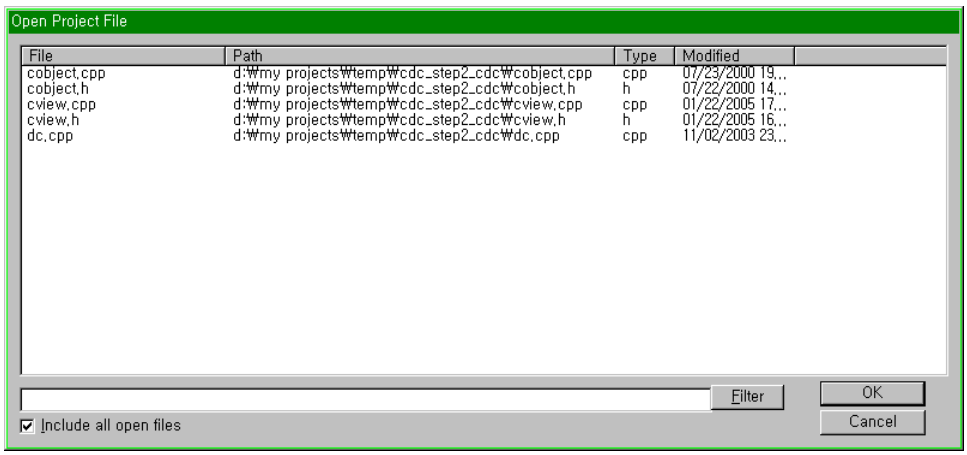
비주얼 어시스트의 멤버 함수 찾아가기 기능은 현재 커서가 위치한 문맥의 클래스의 멤버 함수를 찾아가는 기능을 제공하므로, 빠르게 연관된 멤버 함수로 이동할 수 있다.



[그림 8.23] 비주얼 어시스트의 멤버 함수 찾아가기 기능: 멤버 함수의 이름을 입력하기 시작하면 일치하는 멤버 함수가 강조되고 엔터키를 누르면 해당 멤버 함수로 이동한다.

비주얼 어시스트의 프로젝트 파일 열기 기능도 유용하다. 소스 파일이 클래스의 헤더를 포함한 경우, 위저드 바의 클래스 찾기 기능으로 쉽게 이동할 수 있지만, 전역 변수를 모아둔 파일이나, 이름 공간이 정의된 파일등은 위저드 바의 클래스 찾기 기능으로 쉽게 해당 클래스로 이동할 수 없다. 이 때 비주얼 어시스트의 프로젝트 파일 열기 기능을 선택하고, 파일 이름을 입력하기 시작하면 해당 파일을 강조하고 몇 번의 타이핑으로 해당 파일을 열 수 있다.

혹 독자들은 클래스 뷰나 파일 뷰에서 열려는 파일을 선택하면 되는데, 뭐가 문제지? 라고 생각할 수도 있을 것이다. 하지만, 프로젝트를 이루는 소스 파일의 개수가 1000여개가 된다고 생각해 보라. 긴 파일들의 리스트에서 소스 파일을 찾는 것은 고된 일이지만, 비주얼 어시스트가 있으면 대여섯번의 타이핑만으로 소스를 열수가 있다.

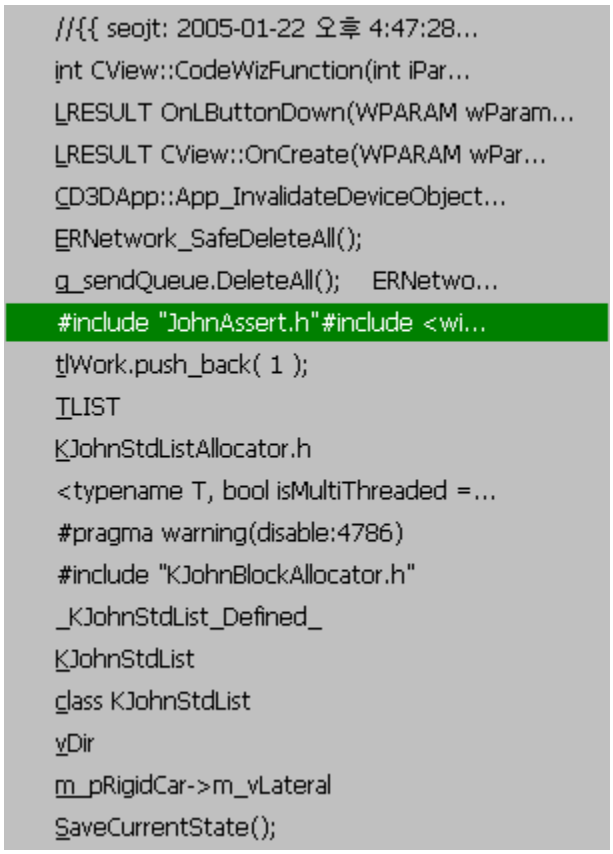


[그림 8.24] 비주얼 어시스트의 프로젝트 파일 열기 기능: 프로젝트 파일 열기를 단축키로 매핑해 놓으면 대여섯번의 타이핑만으로도 소스를 열 수 있다.

비주얼 어시스트의 문맥 표시 기능은 현재 커서가 위치한 곳의 문맥 정보를 동적으로 표시해준다. 예를 들면, 긴 if문에서 복잡한 블록을 편집중일 때, if문의 시작 부분은 화면위로 스크롤 되어 보이지 않고, 특정 블록의 위치가 헷갈릴 때, 비주얼 어시스트의 문맥 기능을 보면 커서가 위치가 곳이 어딘지 쉽게 알 수 있다.

[그림 8.25] 비주얼 어시스트의 문맥 기능:

비주얼 어시스트의 다중 클립보드 기능은 클립 보드로 복사한 최근 텍스트 중에서 하나를 선택하여 붙여넣기를 하는 것을 지원한다. 예를 들면 복사해야 하는 코드가 세 군데에 흩어져 있고, 붙여 넣기를 할 때 이 세 개중 하나를 사용해야 한다고 하자. 비주얼 어시스트가 있다면, 원하는 세 곳의 텍스트를 선택하여 Ctrl+C를 눌러 일반적으로 복사한 다음, 붙여 넣기를 할 때, Ctrl+V가 아니라 Ctrl+Shift+V를 누르면 아래와 같이 클립 보드의 내용을 메뉴로 표시된다. 사용자는 원하는 클립 보드의 내용을 선택하면 된다.



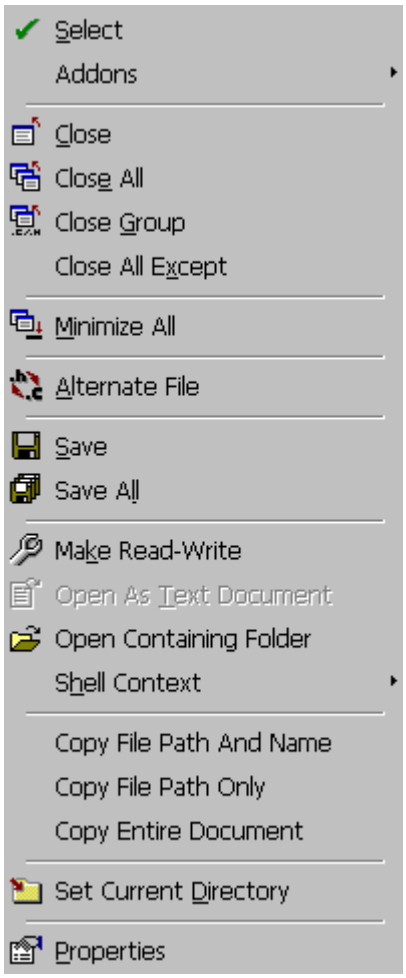
[그림 8.26] 비주얼 어시스트의 다중 클립보드 기능: 클립 보드로 복사되는 최근의 텍스트 중에서 붙여 넣기 원하는 텍스트를 선택할 수 있다.

윈탭(WinTabs)

윈탭은 편집중인 소스 파일을 탭 컨트롤로 표시해 주는 도구이다. 파일 이름을 보고, 열려는 파일을 선택할 수 있으므로 편리하며 다양한 파일 관련 기능들을 제공한다.

[그림 8.27] 윈탭의 파일 탭 기능:

탭 컨트롤 위에서 오른쪽 버튼을 누르면 다양한 종류의 컨텍스트 메뉴를 제공한다. 아래의 그림을 보자.

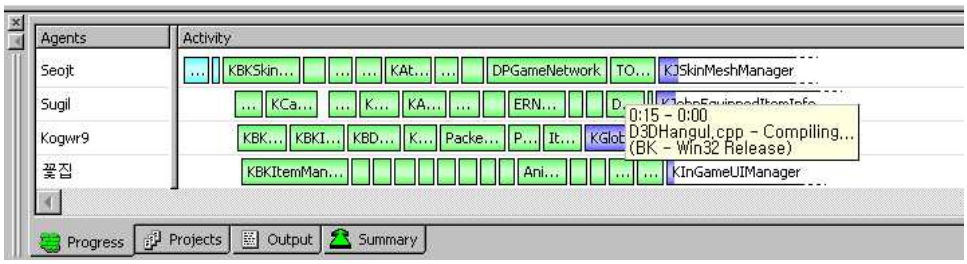


[그림 8.28] 윈탭의 컨텍스트 메뉴:

눈에 띄는 기능은 탐색기의 컨텍스트 메뉴를 그대로 제공한다는 것이다. 그것은 비주얼 C++안에서 탐색기로 할 수 있는 많은 작업을 할 수 있다는 것을 의미한다.

인크레디빌드(IncrediBuild)

인크레디빌드가 없는 비주얼 C++은 생각하기도 싫다! 인크레디빌드는 네트웍을 통하여 분산 컴파일/링크를 지원하는 유용한 툴이다. 필자가 진행 중인 범퍼킹 프로젝트의 경우 Rebuild All을 선택하면 펜티엄4 2.6에서 약 20분이 걸린다. 그런데 인크레디빌드를 사용하면 인크레디빌드 서버에 등록된 모든 컴퓨터와 협조하여 소스를 컴파일하고 링크하여 1분도 걸리지 않아 실행 파일을 생성한다.



[그림 8.29] 인크레디빌드(IncrediBuild)

[문서의 끝]