VIS Laboratory

MFC를 이용한 다이얼로그 기반의 영상 출력 프로그램 작성

VISLAB 박제강

1. 시작하기 전에

영상관련 알고리즘을 개발하는 과정에서 작성한 프로그램을 테스트하고 피드백 하는 작업은 빈번하게 발생한다. 이때 기존 콘솔(Console) 형태로 작성된 프로그램의 경우 테스트 작업을 유동적으로 조절할 수 없기 때문에 작업의 효율이 떨어진다. 반면 GUI(Graphical User Interface) 형태로 작성된 프로그램은 사용자가 자신의 의도를 쉽게 반영하고 그 결과를 빠르게 눈으로 확인할 수 있다는 장점이 있다.

윈도우즈 응용프로그램을 개발하는 방법에는 여러 가지가 있는데 MFC(Microsoft Foundation Class)는 C++클래스 라이브러리 형태로 제공된다. 기존 운영체제가 제공하는 Win32 API(Application Programming Interface) 함수를 직접 사용해서 프로그램을 작성하는 방식의 경우 프로그램을 섬세하게 제어할 수 있고 속도도 빠르지만 방대한 양의 API 함수에 대해서 알아야 하고 작성해야 하는 코드의 양이 많다는 단점이 있다. MFC는 이러한 Win32 API를 클래스로 랩핑(Wrapping)하여 직관적인 이름을 가지는 멤버 함수 형태로 제공하기 때문에 API 함수들을 더 쉽게 사용할 수 있다. 또한 프로그램의 창을 구성하는 각각의 요소(클래스)들의 객체를 생성하고 이를 조립해서 전체 프로그램 창을 구성하기 때문에 사용하기 쉽고 속도도 빠르다는 장점이 있다. 클래스를 사용하기 위한 C++에 대한 지식과 MFC가 제공하는 클래스들의 구조를 일정 수준 이상 이해한다면 간단한 테스트 프로그램 정도는 쉽게 작성할 수 있다.

2. 윈도우즈 응용프로그램 - 메시지와 이벤트

MFC가 Win32 API 함수를 멤버 함수 형태로 제공하기 때문에 Win32 API 함수를 따로 공부할 필요는 없지만 MFC를 사용하더라도 윈도우즈 응용프로그램의 기본적인 특징은 알고 있어야 한다. 그 중에서도 윈도우즈 응용프로그램은 언제 발생할지 알 수 없는 사용자의 입력신호를 메시지라는 개념을 통해 입력 받는다는 점에서 기존의 콘솔 응용프로그램과 차이가 있다.

콘솔 응용프로그램 : 프로그램이 작성 될 때 미리 입력된 일련의 명령들을 순서대로 실행하는 순차적 실행방법을 따른다.

윈도우즈 응용프로그램 : 프로그램의 실행 순서가 정해져 있지 않고 사용자의 입력과 같은 특정 요인(메시지)에 의해 실행 순서가 달라진다.

메시지	설명
WM_CREATE	윈도우가 처음 만들어질 때 발생한다.
WM_DESTROY	윈도우가 메모리에서 삭제될 때 발생한다.
WM_PAINT	화면을 다시 그려야 할 때 발생한다.
WM_LBUTTONDOWN	마우스 왼쪽 버튼을 눌렀을 때 발생한다.
WM_MOUSEWHEEL	마우스 휠을 돌렸을 때 발생한다.
WM_KEYDOWN	키보드의 시스템 키를 제외한 나머지 키를 눌렀을 때 발생한다.

표 1. 자주 사용되는 윈도우즈 메시지

표1에서 알 수 있듯이 메시지에는 여러 가지 종류가 있지만 메시지의 개념을 이해하기 가장 좋은 예는 WM_KEYDOWN이나 WM_LBUTTONDOWN과 같은 키보드나 마우스의 입력과 관련된 메시지이다. 사용자가 키보드의 어느 버튼을 언제 누를지는 프로그램을 작성할 때 미리 알 수 없는 사항이다. 때문에 윈도우즈 프로그램은 사용자가 언제 어떤 버튼을 누르는지를 항상 체크하고 있어야 한다.

윈도우즈 프로그램은 이와 같은 사용자나 시스템의 내부적인 동작에 의해 발생된 변화에 대한 정보를 메시지라는 개념으로 표현한다. 또한 발생된 메시지들을 순서대로 저장해 두는 공간을 메시지 큐라고 하고 메시지 큐에 저장된 메시지를 꺼내서 처리하는 부분을 메시지 루프라고 한다. 메시지가 처리되는 구조는 비쥬얼 스튜디오에서 윈도우즈 응용프로그램 프로젝트를 생성하면 미리 작성되어 있기 때문에 프로그래머가 직접 구현할 필요는 없다. 프로그램 작성자는 특정 메시지가 발생했을 때 어떤 작업을 수행할 것인지를 구현하면 된다.

그림 1. 클래스 속성창의 메시지 탭(왼쪽), 메시지 맵(오른쪽 위), 메시지와 연결된 함수(오른쪽 아래)

그림1은 비쥬얼 스튜디오에서 메시지를 등록하거나 편집할 수 있는 클래스 속성창의 메시지 탭(왼쪽)과 실제 코드에 메시지와 이벤트가 등록되는 메시지 맵(오른쪽 위), 그리고 해당 메시지가 발생하면 실행되는 함수(오른쪽 아래)를 보여주고 있다. 메시지 맵에 등록된 메시지가 발생하면 그에 해당하는 함수가 실행되므로 프로그래머는 수행할 작업 내용을 함수 내부에 구현하면 된다. 메시지를 등록할 때 프로그램 작성자가 윈도우즈 응용프로그램 구조에 익숙하다면 코드를 직접 수정하여 메시지를 등록할 수도 있지만 그렇지 않다면 클래스 속성창의 메시지 탭(왼쪽)을 통해 메시지를 등록하는 것이 일반적이다.

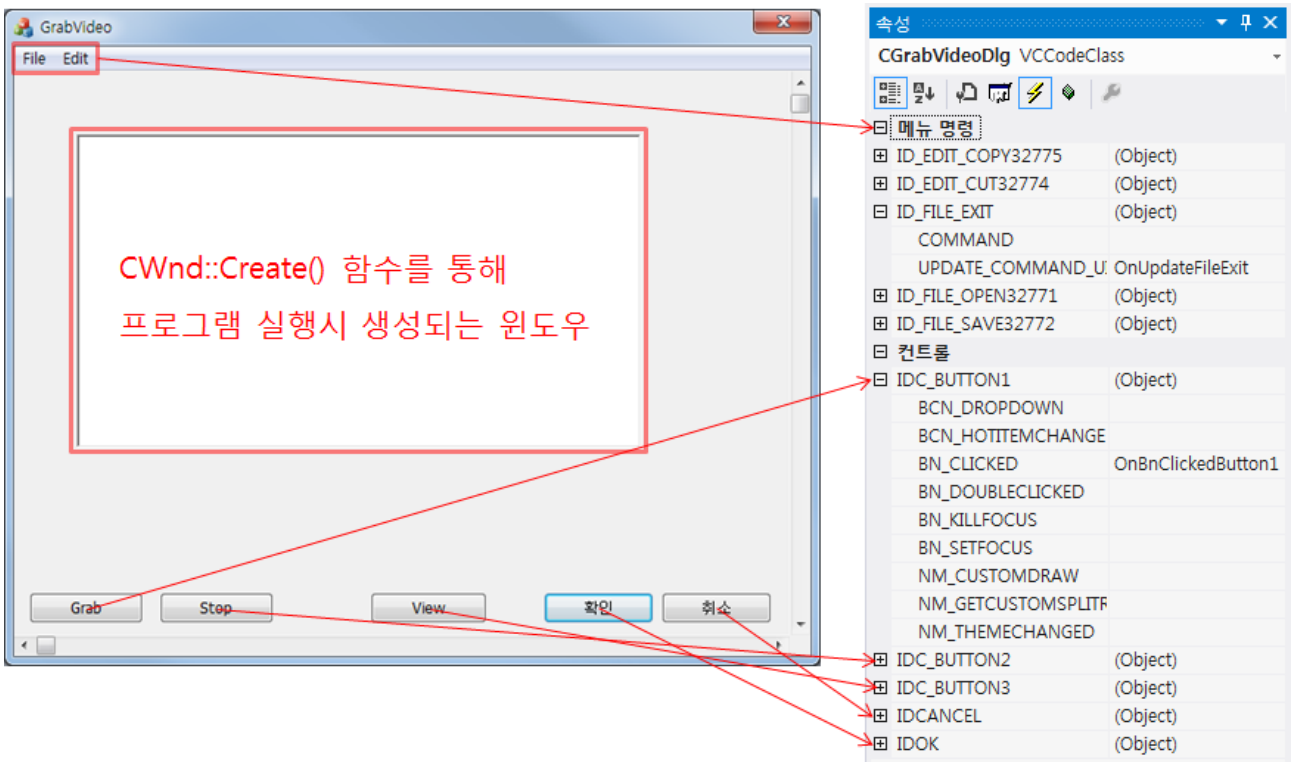


그림 2. 다이얼로그 기반 프로그램의 실행화면(왼쪽), 클래스 속성창의 이벤트 탭(오른쪽)

또한 그림1의 메시지 맵을 보면 WM_로 시작하는 메시지 외에 다른 메시지들이 있다. 이러한 메시지들은 이벤트라고 불리며 리소스 창에서 미리 생성한 메뉴나 커맨드 버튼과 같은 객체에서 발생한 메시지를 의미한다. 그림2는 프로그램이 실행된 화면(왼쪽)과 클래스 속성창의 이벤트 탭(오른쪽)을 보여주고 있다. 리소스 창에서 생성한 메뉴와 커맨드 버튼 객체가 나열되어 있고 하위 메뉴에서는 해당 객체에 대해 등록하거나 편집 할 수 있는 메시지를 확인할 수 있다. 앞서 설명하였듯이 클래스 속성창의 이벤트 탭에서는 리소스 창에서 미리 생성한 객체만 확인 할 수 있다. 그림2의 다이얼로그 창에서 중앙의 흰색 윈도우는 프로그램이 실행된 후에 윈도우 생성 명령어를 통해 생성되는 객체이다. 때문에 프로그램이 실행되기 전 시점인 속성 창을 통해서는 메시지를 등록할 수 없고 메시지를 등록하려면 코드를 직접 추가하여 등록해야 한다.

실제로 다이얼로그 기반의 MFC 프로젝트를 생성하면 알 수 없는 코드로 가득한 소스파일들이 생성되어 당황하는 경우가 많다. 하지만 응용프로그램 작성자는 메시지와 관련된 일부 함수만 알아도 간단한 테스트 프로그램을 작성하기에는 전혀 문제가 없다. 문서 마지막의 다이얼로그 기반의 프로젝트를 생성하는 예제를 통해 이러한 내용을 확인할 수 있다.

3. MFC의 구조

MFC는 C++ 클래스에 대해서 공부하기 가장 좋은 예제이다. 클래스를 공부하다 보면 생성자와 소멸자, 상속, 다형성과 같은 내용을 접하게 되는데 책에 포함된 간단한 예제만 가지고 이러한 내용들이 어디에 어떻게 사용되는지 감을 잡기는 어렵다. 심지어 구조체 대신 클래스를 사용하는 이유조차도 이해하기 쉽지 않다. MFC는 이러한 클래스의 특징들을 굉장히 잘 활용하고 있다.

1) CWnd 클래스

CObject 클래스는 MFC의 기본적인 서비스를 포함하는 기반 클래스이다. MFC 클래스들은 CObject 클래스를 상속받는 클래스와 상속받지 않는 클래스로 분류된다. 전체 MFC 클래스 계층도는 http://msdn.microsoft.com/ko-kr/library/ws8s10w4.aspx에서 확인할 수 있다.

이렇게 방대한 MFC 클래스 중 테스트 프로그램 작성자가 주의 깊게 살펴봐야 하는 클래스는 CWnd 클래스이다. CWnd 클래스는 사용자의 눈에 보이는 윈도우 창의 기본 기능을 제공하는 클래스이다. 쉽게 말해서 CWnd 객체를 생성한다는 것은 윈도우 창의 생성을 의미한다. 프로그램 작성자는 이 CWnd 클래스를 상속받아서 특정 기능을 수행하는 자신만의 윈도우 클래스를 만들 수 있다. MFC는 컨트롤 객체(버튼, 콤보박스 등)와 다이얼로그, 프레임 윈도우, 뷰와 같이 일반적으로 사용되는 클래스를 기본적으로 제공하는데 이런 클래스들은 모두 CWnd 클래스를 상속받아서 만들어진다.

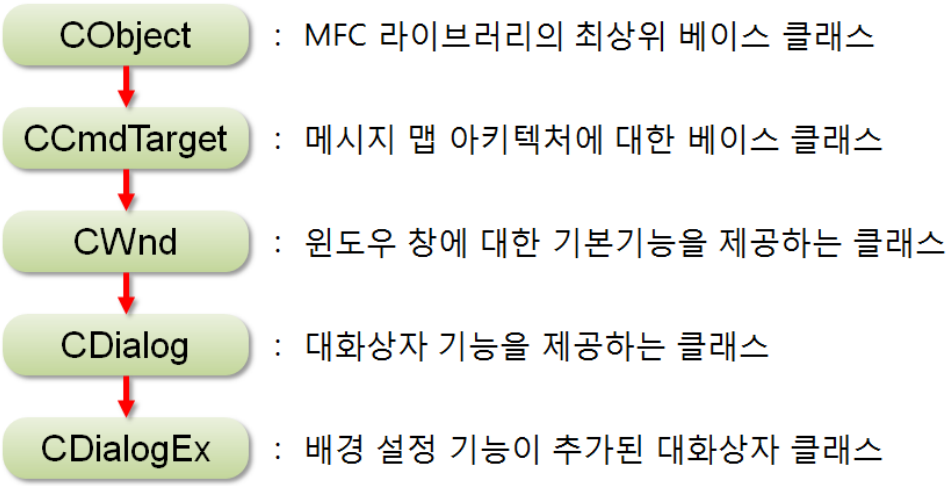


그림 3. CDialogEx 클래스의 상속관계

그림3은 대화상자기반 윈도우 생성 시 기본 클래스로 사용되는 CDialogEx 클래스의 상속관계를 나타낸다. 대화상자도 일단 윈도우 창의 한 종류이기 때문에 CWnd 클래스를 상속하였고 대화상자의 기본 기능을 포함하고 있는 CDialog 클래스도 상속하였다. CDialogEx 클래스는 대화상자의 배경색과 배경화면을 설정하는 기능이 추가된 클래스로 MFC 라이브러리 9.0 버전 이상에서 사용 가능하다. 이처럼 MFC는 일반적으로 사용되는 윈도우 클래스를 제공하고 프로그램 작성자는 이를 상속하여 자신만의 윈도우 클래스를 만들 수 있다.

2) 윈도우 인스턴스의 부모-자식 관계

앞장에서 설명하였듯이 MFC에서 화면에 보이는 윈도우 클래스(버튼, 콤보박스, 뷰 등)들은 모두 CWnd 클래스를 상속받아 만들어진다고 하였다. 그리고 실제 프로그램의 화면은 윈도우 클래스의 인스턴스(클래스가 메모리에 실제로 구현된 실체를 의미하는 말로 흔히 표현하는 객체라고 생각해도 무관하다)들로 구성된다.

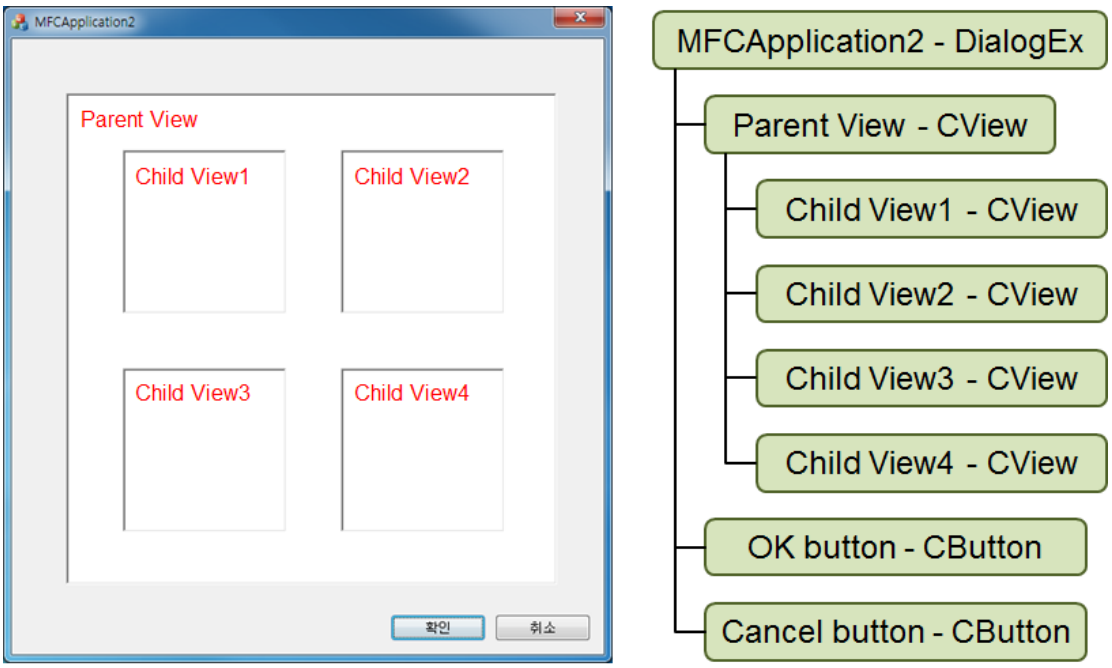


그림 4. MFC 프로그램을 구성하는 각 윈도우 인스턴스들의 부모-자식 관계도

그림 4의 다이얼로그 기반 프로그램은 총 8개의 윈도우 인스턴스들로 구성된다. 각각 다이얼로그 인스턴스 1개와 CView 인스턴스 5개, CButton 인스턴스 2개 이다. 이 인스턴스들은 그림 4의 관계도(오른쪽)에서 보이듯이 부모-자식 관계를 가지고 있다. 먼저 CView 인스턴스들 중에서 다른 CView 인스턴스들을 포함하는 가장 큰 CView 인스턴스(Parent View)와 2개의 CButton 인스턴스들은 DialogEx 인스턴스를 부모로 가진다. 그리고 나머지 4개의 작은 CView 인스턴스(Child View)들은 큰 CView 인스턴스(Parent View)를 부모로 가진다.

이렇게 윈도우 인스턴스들이 부모-자식 관계로 구성되기 때문에 가지는 몇 가지 특징들이 있다. 그 중 가장 큰 특징은 부모 인스턴스의 몇 가지 속성이 자식 인스턴스에도 함께 적용된다는 점이다. 일반적으로 자식 인스턴스는 부모 인스턴스 위에 그려지게 되고 자식 인스턴스는 부모 인스턴스를 기준으로 하는 좌표계 위에 위치하기 때문에 부모 인스턴스가 움직이면 자식 인스턴스도 같이 움직인다. 마찬가지로 부모 인스턴스가 삭제되면 자식 인스턴스도 함께 삭제된다. 하지만 모든 속성을 부모 인스턴스와 자식 인스턴스가 공유하지는 않는다. 예를 들어 윈도우 바탕색이나 커서의 모양 등은 각자 독립된 속성 값을 가진다.

그림 4에서 DialogEx 인스턴스는 다른 인스턴스들을 자식으로 가지는 최상위 인스턴스이다. 이렇게 자신 스스로가 하나의 독립된 윈도우를 만드는 인스턴스는 NULL 값을 부모로 가지게 된다. 프로그램 실행 시 생성되는 기본 윈도우 창이 아닌 독립된 새로운 윈도우 창을 생성하고 싶다면 윈도우 인스턴스 생성 함수

의 인자 중 CWnd* 형을 가지는 부모 인스턴스 주소값에 NULL값을 넘겨주면 된다.

4. 마치며

지금까지 설명한 내용은 MFC를 이용해서 윈도우즈 응용프로그램을 만드는데 필요한 굉장히 기본적인 부분이다. 실제로 여기에서 설명한 내용으로는 다이얼로그 기반의 간단한 테스트 프로그램을 작성하는 것이 고작일 것이다. 만약 윈도우즈 응용프로그램에 관심이 있다면 MFC를 공부하기 보다는 Windows API를 먼저 공부해야 한다. 기존의 윈도우즈 응용프로그래머에게 MFC는 굉장히 편리한 도구가 될지 모르지만 윈도우즈 응용프로그램을 처음 배우는 사람에게는 단지 API의 방대한 함수들을 랩핑하고 있는 껍데기에 불과하다. 처음 MFC를 공부할 때 예제를 따라서 뭔가 만들긴 했는데 작성한 코드를 실행하면 왜 이런 윈도우가 생성되는지 이해하기 어려운 이유가 이 때문이다. 이 문서에 포함된 예제도 마찬가지이다. 윈도우가 어떻게 생성되고 메시지를 어떻게 처리하는지 등의 기본적이 내용이 궁금하다면 Windows API에 관련된 책의 앞부분을 보길 바란다.

예제 - 다이얼로그 기반 프로젝트에서 MyScrollView 클래스 작성 및 테스트

목표 : 이미지 디스플레이와 스크롤바 기능을 지원하는 MyScrollView 클래스 작성.

핵심 : MFC에서 제공하는 CScrollView 클래스는 스크롤바와 관련된 기능들을 기본적으로 포함하고 있다. CScrollView 클래스를 상속받으면 이런 기능들의 대부분을 그대로 사용할 수 있다.

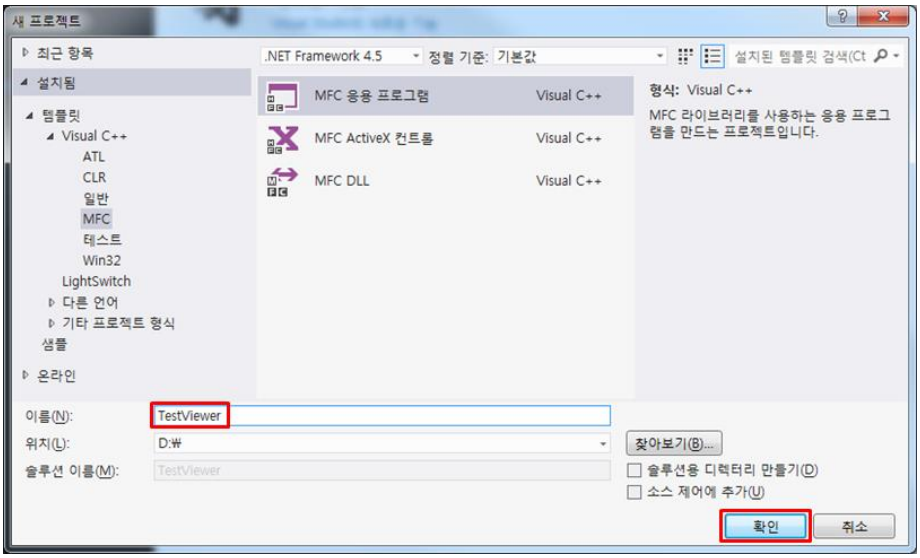
환경 : Microsoft Visual Studio Professional 2012

과정 :

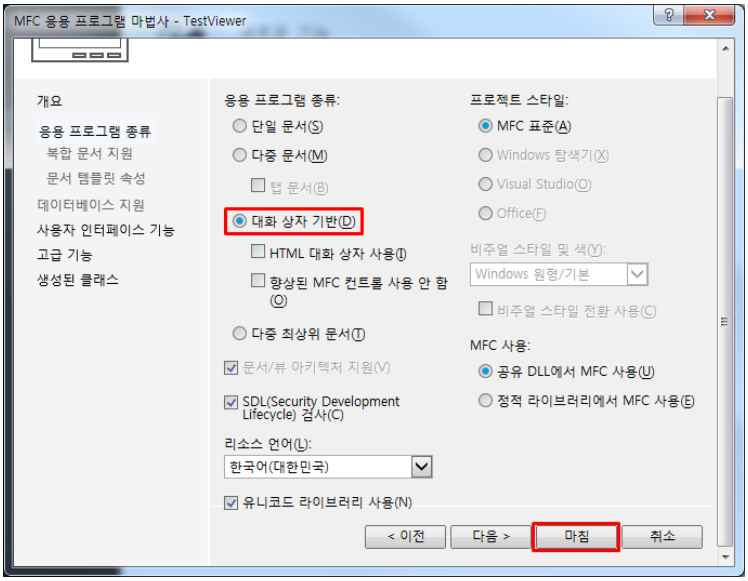
1. 다이얼로그 프로젝트 생성

- 템플릿 : MFC - MFC 응용 프로그램

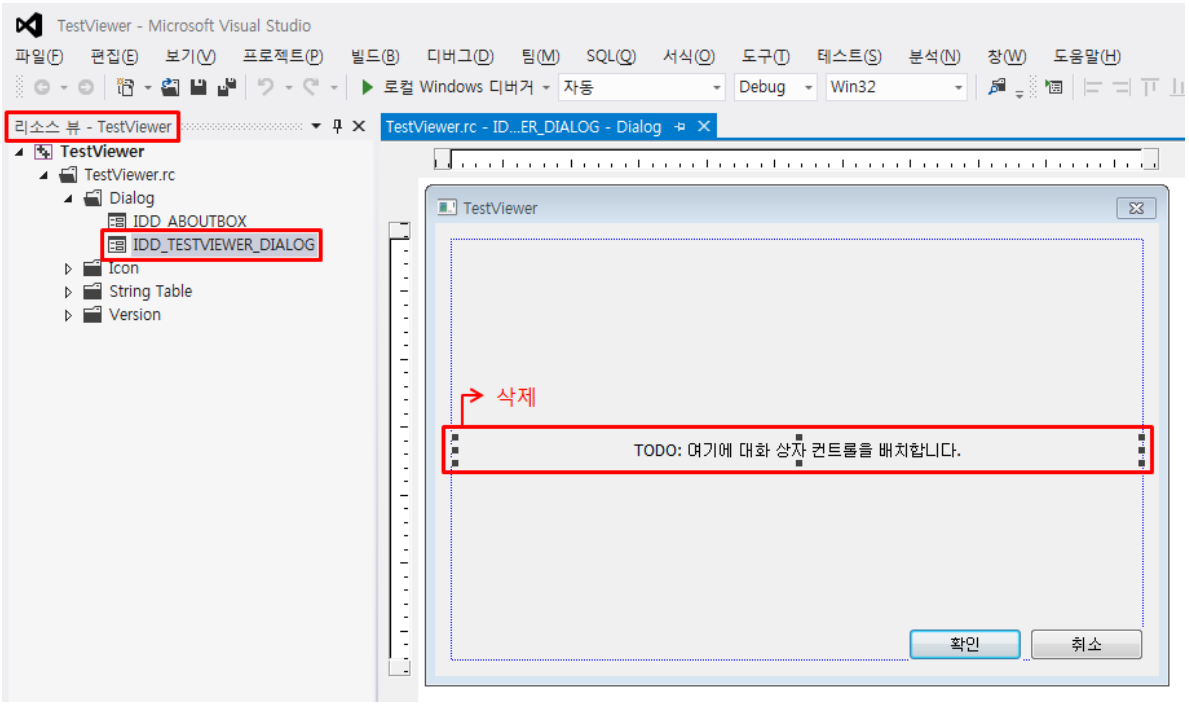
  프로젝트 이름 : TestViewer



- 응용 프로그램 종류 : 대화 상자 기반



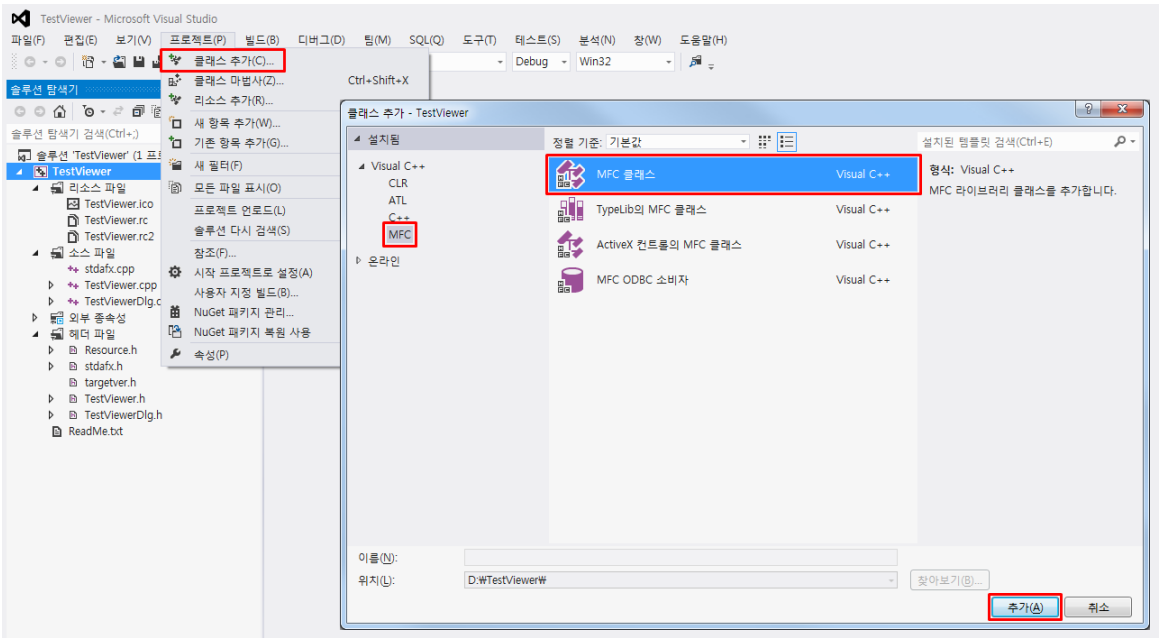
- 기본 텍스트 컨트롤 삭제



· 안내를 위한 일종의 주석 같은 의미로 삽입되어 있는 텍스트 컨트롤이므로 삭제한다.

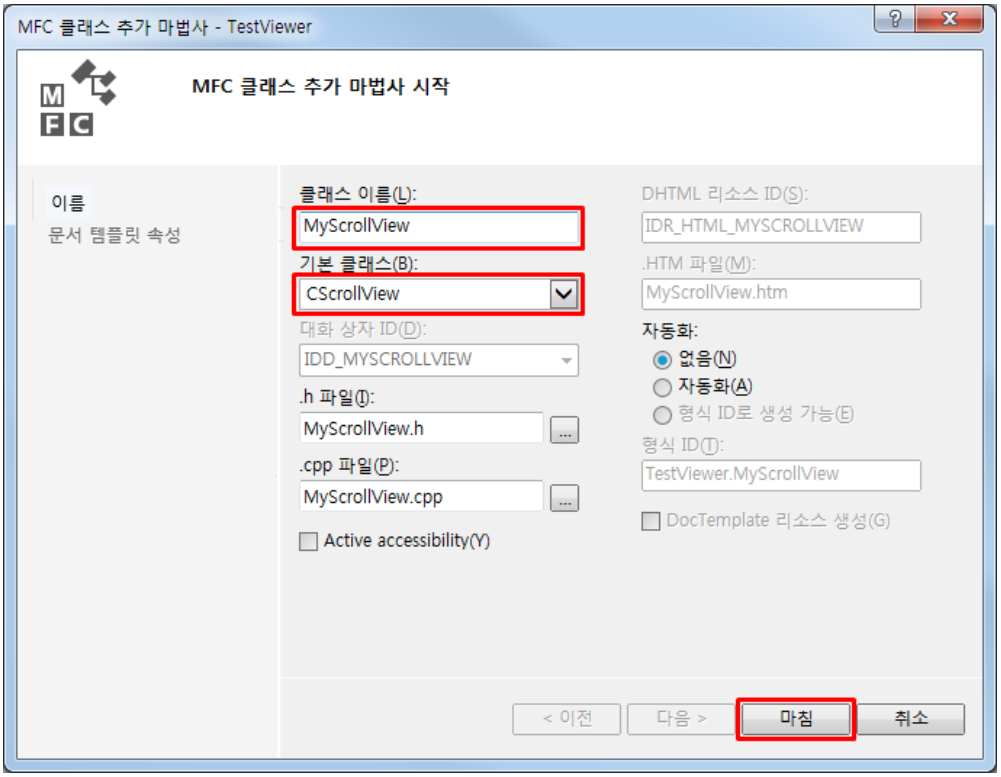
2. MyScrollView 클래스 작성

- 클래스 추가 : MFC - MFC클래스

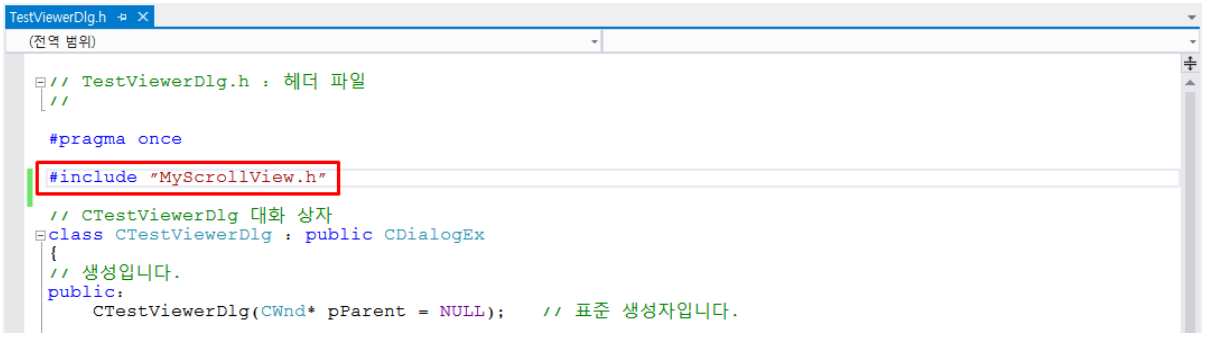


클래스 이름 : MyScrollView

기본 클래스 : CScrollView

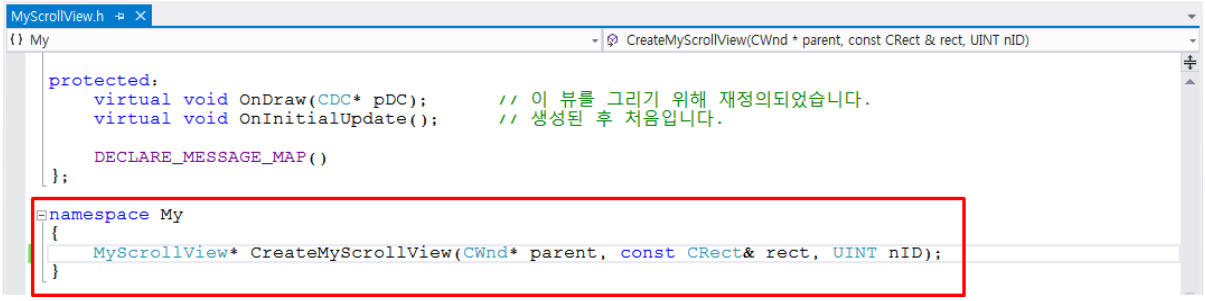


- TestViewerDlg.h에 MyScrollView.h 포함



· 최상위 인스턴스인 다이얼로그 창에 MyScrollView 인스턴스를 생성하기 위해 헤더파일을 포함한다.

- MyScrollView 클래스의 인스턴스 생성함수 선언 및 구현



· MyScrollView.h 파일의 MyScrollView 클래스 선언 아래쪽에 인스턴스 생성함수를 선언한다.

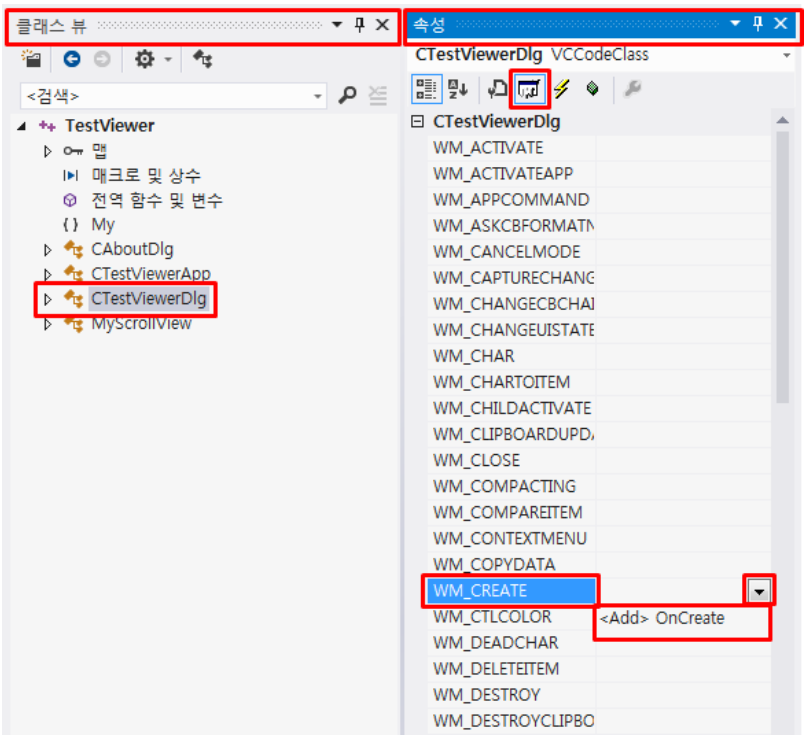
```cpp
// MyScrollView.cpp : 구현 파일입니다.
//

#include "stdafx.h"
#include "TestViewer.h"
#include "MyScrollView.h"

// My::CreateMyScrollView

MyScrollView* My::CreateMyScrollView(CWnd* parent, const CRect& rect, UINT nID)
{
    CRuntimeClass *pObject;
    pObject = RUNTIME_CLASS(MyScrollView);
    MyScrollView* view = (MyScrollView*) pObject->CreateObject();
    if (!view->Create(NULL, NULL, AFX_WS_DEFAULT_VIEW, rect, parent, nID, NULL))
    {
        TRACE0("Failed to create MyScrollView window\n");
        return 0;
    }
    return view;
}
```

· MyScrollView.cpp 파일에 인스턴스 생성함수를 구현한다.

· MFC에서 제공하는 기본 클래스들 중에서 몇몇 클래스는 기본 생성자를 통해 객체를 생성할 수 없다. CScrollView의 상위 클래스인 CView가 이에 해당한다. (보통 생성자가 protected 형태로 선언되어 외부에서 접근할 수 없게 되어있다. Frame이나 Document 클래스의 유도 클래스도 이에 해당한다.) 이런 경우 일반적인 방법으로 객체를 생성할 수 없고 런타임 클래스에 해당 클래스를 등록 후 객체를 생성해야 한다. MSDN에 런타임 클래스에 대한 자세한 내용이 설명되어 있다.

3. 중간 테스트

- CTestViewerDlg 클래스에 WM_CREATE 메시지 추가



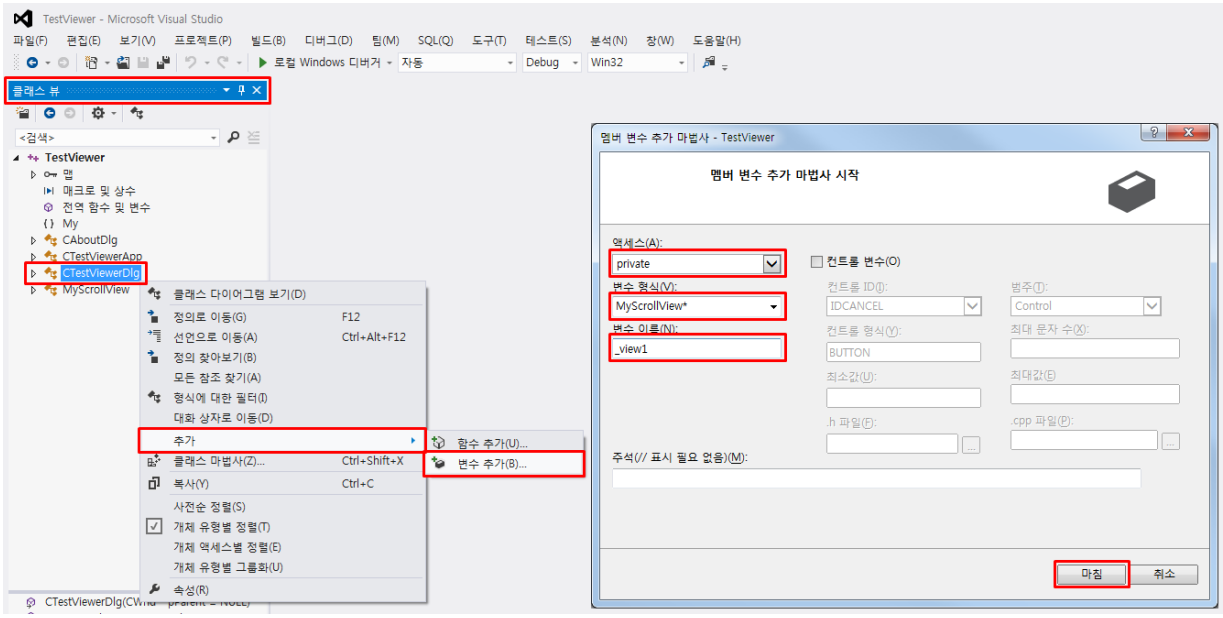
· 속성창의 메시지 등록 기능을 통해 메시지를 등록하면 함수 및 메시지 맵에 메시지를 등록하는 코드가 자동으로 삽입된다.

- MyScrollView 포인터 변수 선언

엑세스 : private

변수 형식 : MyScrollView*

변수 이름 : _view1



· 클래스 뷰 - CTestViewerDlg 클래스 오른쪽 클릭 - 추가 - 변수 추가

· 코드 상에서 직접 변수를 추가하지 않고 마법사를 사용할 경우 클래스 생성자에 변수를 초기화 하는 코드가 자동으로 추가된다.

- MyScrollView 인스턴스 생성

TestViewerDlg.cpp 파일의 CTestViewerDlg::OnCreate() 함수의 구현부에 인스턴스 생성함수를 호출하는 코드를 추가한다.

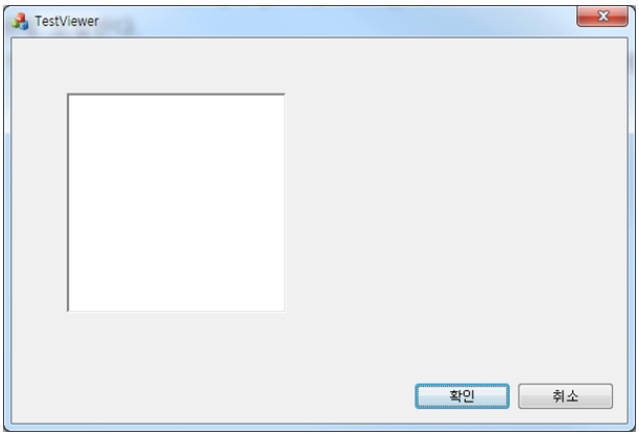
```
TestViewerDlg.cpp
CTestViewerDlg
OnCreate(LPCREATESTRUCT lpCreateStruct)

int CTestViewerDlg::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CDialogEx::OnCreate(lpCreateStruct) == -1)
        return -1;

    // TODO:  여기에 특수화된 작성 코드를 추가합니다.
    _view1 = My::CreateMyScrollView(this, CRect(50, 50, 250, 250), 1000);

    return 0;
}
```

· 최상위 인스턴스인 다이얼로그 창 위에 MyScrollView 인스턴스를 생성하기 위해 부모 인스턴스의 주소값 인자는 this를 넘겨준다. (AfxGetMainWnd() 함수를 사용해도 된다.)

· 생성되는 윈도우의 크기는 적절하게 조절한다.

· 인스턴스의 ID를 의미하는 nID 인자는 리소스 창에서 생성한 컨트롤 객체의 ID와 겹치지 않게 적당한 값을 사용한다. (Resource.h 파일 참조)

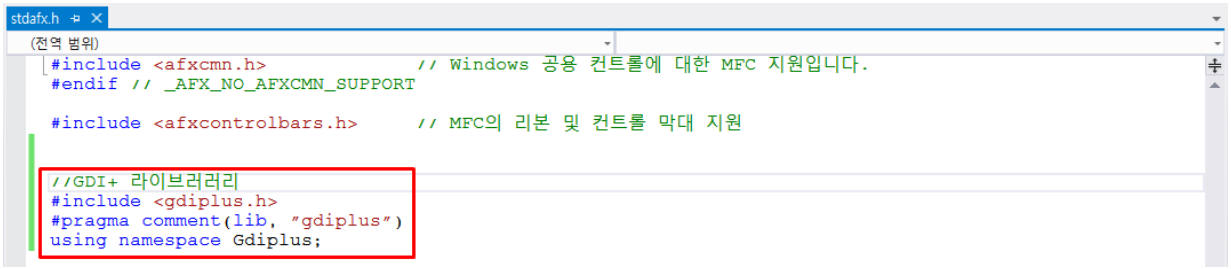
- 빌드 및 실행



· MyScrollView 인스턴스의 창 크기에 비해 가로 및 세로 스크롤바의 크기가 작기 때문에 아직 화면에 스크롤바가 표시되지 않는다.

4. MyScrollView에 이미지 그리기

- GDI+ 라이브러리 추가



· stdafx.h 파일에 선언하면 프로젝트의 다른 파일에서도 GDI+ 라이브러리를 사용할 수 있다.

· GDI는 윈도우즈에서 제공하는 그래픽 출력 라이브러리로 화면에 선을 긋거나 이미지를 출력하는 등의 그래픽 작업을 지원한다. GDI+는 윈도우즈 XP 이상에서 제공하는 기존의 GDI보다 그래픽 및 객체화에 대한 부분이 개선된 라이브러리이다.

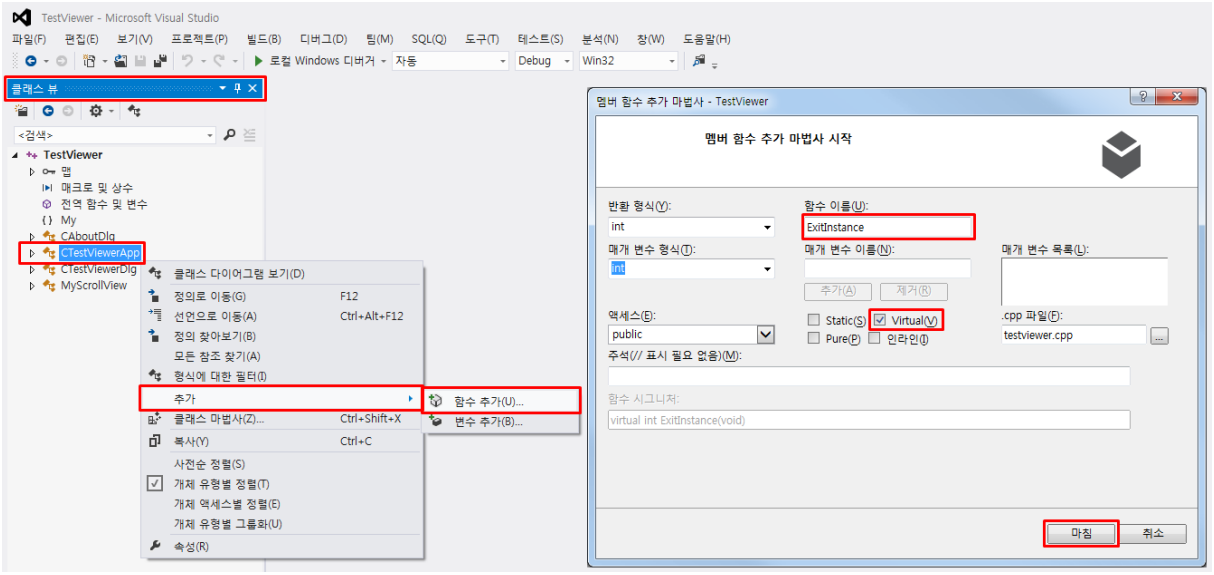
- GDI+ 라이브러리 초기화

```
TestViewer.cpp
CTestViewerApp                                  InitInstance()

// 유일한 CTestViewerApp 개체입니다.

CTestViewerApp theApp;

ULONG_PTR gdiplusToken;          //GDI+ 토큰

// CTestViewerApp 초기화

BOOL CTestViewerApp::InitInstance()
{
    // 응용 프로그램 매니페스트가 ComCtl32.dll 버전 6 이상을 사용하여 비주얼 스타일을
    // 사용하도록 지정하는 경우, Windows XP 상에서 반드시 InitCommonControlsEx()가 필요합니다.
    // InitCommonControlsEx()를 사용하지 않으면 창을 만들 수 없습니다.
    INITCOMMONCONTROLSEX InitCtrls;
    InitCtrls.dwSize = sizeof(InitCtrls);
    // 응용 프로그램에서 사용할 모든 공용 컨트롤 클래스를 포함하도록
    // 이 항목을 설정하십시오.
    InitCtrls.dwICC = ICC_WIN95_CLASSES;
    InitCommonControlsEx(&InitCtrls);

    CWinApp::InitInstance();

    //GDI+ 초기화
    GdiplusStartupInput gdiplusStartupInput;
    if(::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL) != Ok)
    {
        AfxMessageBox(TEXT("ERROR:Failed to initialize GDI+ library!"));
        return FALSE;
    }
```

· TestViewer.cpp 에서 작업한다.

· GDI+ 라이브러리를 사용하기 위해서는 반드시 초기화 단계가 필요하다.

- ExitInstance() 함수 재정의



· 다이얼로그 기반 프로젝트는 프로젝트 생성 시 ExitInstance() 함수의 재정의가 포함되어있지 않기 때문에 직접 추가해준다.

- GDI+ 라이브러리 해제

```
TestViewer.cpp
CTestViewerApp                          ExitInstance(void)

int CTestViewerApp::ExitInstance(void)
{
    //GDI+ 라이브러리 해제
    ::GdiplusShutdown(gdiplusToken);

    return CWinApp::ExitInstance();
}
```

· 재정의한 ExitInstance() 함수에 GDI+ 라이브러리를 해제하는 구분을 추가한다. GDI+ 초기화와 마찬가지로 꼭 필요한 단계이다. 여기까지 GDI+ 라이브러리를 사용하기 위한 준비단계이다.

- 이미지 렌더링

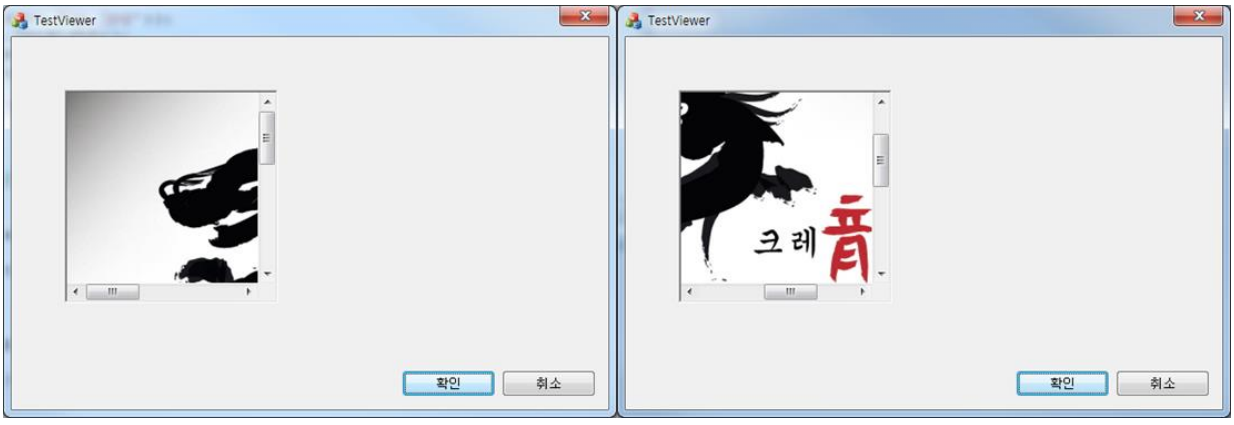
```
MyScrollView.cpp
MyScrollView                            OnDraw(CDC * pDC)

void MyScrollView::OnDraw(CDC* pDC)
{
    CDocument* pDoc = GetDocument();
    // TODO: 여기에 그리기 코드를 추가합니다.

    Graphics graphics(pDC->GetSafeHdc());

    Image image(_T("Crayon Pop.jpg"));
    UINT width = image.GetWidth();
    UINT height = image.GetHeight();
    graphics.DrawImage(&image, Rect(0, 0, width, height));

    SetScrollSizes(MM_TEXT, CSize(width, height));
}
```

· 이미지 파일로부터 데이터를 읽어오고 화면에 그려준다.

· SetScrollSizes() 함수를 통해 스크롤바의 크기를 이미지의 크기와 같아지게 설정한다.

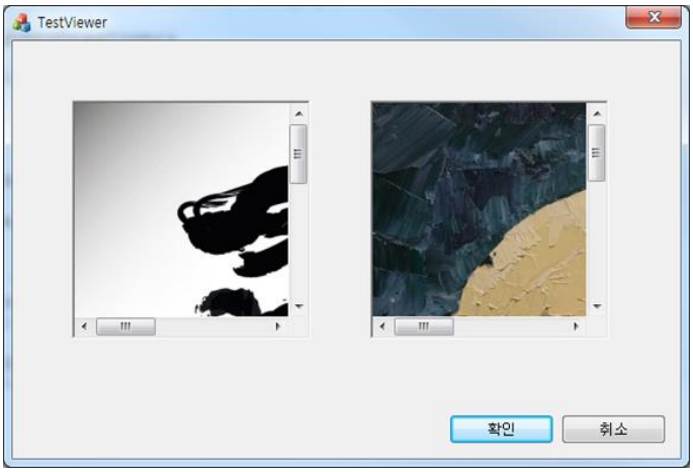
4. 최종 테스트

- 빌드 및 실행



· MyScrollView 인스턴스 창에 이미지가 렌더링 되고 스크롤바를 이용해 이미지 전체를 살펴볼 수 있다. 직접 CWnd 클래스를 상속받아서 스크롤바 기능을 지원하는 윈도우 클래스를 만들기 보다는 MFC에서 제공하는 기본 클래스를 상속받아서 만드는 것이 훨씬 효율적이다.

- MyScrollView 인스턴스 추가



· My::CreateMyScrollView() 인스턴스 생성함수를 한번 더 호출하여 인스턴스 창을 하나더 추가할 수 있다. MyScrollView 클래스를 작성했던 앞의 과정을 이해하고 있다면 클래스를 조금 수정하여 위의 그림처럼 각각의 창이 다른 이미지를 렌더링 하도록 만들 수 있을 것이다.

· 지금까지 작성한 MyScrollView 클래스는 아직 보완할 점이 많다. 예를 들면 마우스 휠을 돌려서 스크롤하는 기능 등은 작동하지 않는다. 이런 MFC의 기능과 관련된 부분은 MFC 관련 서적을 조금만 공부하면 충분히 작성할 수 있을 것이다.

References

[1] 최호성, 열혈강의 Visual C++ 2008 MFC 윈도우 프로그래밍, FREELEC, 2009.

[2] 김상형, Windows API정복, 가남사, 2001.

[3] MFC 프로그래밍 연구회, Visual C++ 2010 MFC 프로그래밍, 세진북스, 2011.