|  |  |  |
| --- | --- | --- |
| 아래의 WTL의 설명은 본인이 일본의 <http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/index.html> 에서 가져온 자료를 번역기로 1차 번역 후 본인이 약간 손을 본 자료입니다. WTL에 대해서 좋은 정보를 얻으셨다면 위 사이트의 주인의 덕택입니다.   |  |  | | --- | --- | |  |  | |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  |  |  | ATL/WTL | |
|  |

|  |  |
| --- | --- |
|  | 처음 |
| * [시작](http://jacking75.cafe24.com/WTL/1.htm) * [WTL의 인스톨](http://jacking75.cafe24.com/WTL/2.htm) | |

|  |  |
| --- | --- |
|  | Hello, ATL/WTL |
| * [윈도우](http://jacking75.cafe24.com/WTL/3.htm) * [윈도우 클래스 정보](http://jacking75.cafe24.com/WTL/4.htm) * [윈도우 특성](http://jacking75.cafe24.com/WTL/5.htm) | |

|  |  |
| --- | --- |
|  | 메시지 |
| * [ATL의 메세지 맵](http://jacking75.cafe24.com/WTL/6.htm) * [WTL의 메세지 맵](http://jacking75.cafe24.com/WTL/7.htm) * [체인](http://jacking75.cafe24.com/WTL/8.htm) * [대체 메세지 맵](http://jacking75.cafe24.com/WTL/9.htm) * [메세지 루프](http://jacking75.cafe24.com/WTL/10.htm) * [메시지 필터](http://jacking75.cafe24.com/WTL/11.htm) * [아이들 핸들러](http://jacking75.cafe24.com/WTL/12.htm) | |

|  |  |
| --- | --- |
|  | GDI |
| * [디바이스 컨텍스트](http://jacking75.cafe24.com/WTL/13.htm) * [펜](http://jacking75.cafe24.com/WTL/14.htm) * [브러쉬](http://jacking75.cafe24.com/WTL/14.htm) * [폰트](http://jacking75.cafe24.com/WTL/15.htm) * [팔렛트](http://jacking75.cafe24.com/WTL/16.htm) * [비트맵](http://jacking75.cafe24.com/WTL/17.htm) * [리젼](http://jacking75.cafe24.com/WTL/18.htm) * [확장 메타 파일](http://jacking75.cafe24.com/WTL/19.htm) | |

|  |  |
| --- | --- |
|  | 메뉴 |
| * [메뉴 바](http://jacking75.cafe24.com/WTL/20.htm) * [팝업 메뉴](http://jacking75.cafe24.com/WTL/21.htm) | |

|  |  |
| --- | --- |
|  | 프레임 윈도우 |
| * [프레임 윈도우](http://jacking75.cafe24.com/WTL/22.htm) * [프레임 윈도우 클래스 정보](http://jacking75.cafe24.com/WTL/23.htm) * [심플 툴바](http://jacking75.cafe24.com/WTL/24.htm) * [심플 리바](http://jacking75.cafe24.com/WTL/25.htm) * [심플 스테이터스 바](http://jacking75.cafe24.com/WTL/26.htm) * [UI갱신 핸들러 – 팝업 메뉴 아이템](http://jacking75.cafe24.com/WTL/27.htm) * [UI갱신 핸들러 – 툴바](http://jacking75.cafe24.com/WTL/28.htm) * [뷰 윈도우](http://jacking75.cafe24.com/WTL/29.htm) * [커맨드 체인](http://jacking75.cafe24.com/WTL/30.htm) | |

|  |  |
| --- | --- |
|  | 다이얼로그 |
| * [모달 다이얼로그](http://jacking75.cafe24.com/WTL/31.htm) * [모달리스 다이얼로그](http://jacking75.cafe24.com/WTL/32.htm) * [UI갱신 핸들러 – 다이얼로그의 팝업 매뉴 아이템](http://jacking75.cafe24.com/WTL/33.htm) * [UI갱신 핸들러 – 자식 윈도우](http://jacking75.cafe24.com/WTL/34.htm) * [키보드 액셀레이터](http://jacking75.cafe24.com/WTL/35.htm) * [심플 다이얼로그](http://jacking75.cafe24.com/WTL/36.htm) | |

|  |  |
| --- | --- |
|  | 위자드 |
| * [ATL/WTL AppWizard](http://jacking75.cafe24.com/WTL/37.html) * [스켈톤](http://jacking75.cafe24.com/WTL/38.htm) * [\_ATL\_MIN\_CRT에 대해서](http://jacking75.cafe24.com/WTL/39.html) | |

|  |  |
| --- | --- |
|  | 표준 컨트룰러 |
| * [버튼](http://jacking75.cafe24.com/WTL/40.htm) * [스태틱](http://jacking75.cafe24.com/WTL/41.htm) * [리스트 박스](http://jacking75.cafe24.com/WTL/42.htm) * [콤보 박스](http://jacking75.cafe24.com/WTL/43.htm) * [스크롤 바](http://jacking75.cafe24.com/WTL/44.htm) * [에디터](http://jacking75.cafe24.com/WTL/45.htm) | |

|  |  |
| --- | --- |
|  | 공용 컨트룰러 |
| * [툴팁](http://jacking75.cafe24.com/WTL/46.htm) * [풍선 툴팁](http://jacking75.cafe24.com/WTL/47.htm) * [리스트뷰](http://jacking75.cafe24.com/WTL/48.htm) * [이미지 리스트](http://jacking75.cafe24.com/WTL/49.htm) * [헤더](http://jacking75.cafe24.com/WTL/50.htm) * [트리 뷰](http://jacking75.cafe24.com/WTL/51.htm) * [확장 트리 뷰](http://jacking75.cafe24.com/WTL/52.htm) * [탭](http://jacking75.cafe24.com/WTL/53.htm) * [리치 에디터](http://jacking75.cafe24.com/WTL/54.htm) * [트랙바](http://jacking75.cafe24.com/WTL/55.htm) * [업 다운](http://jacking75.cafe24.com/WTL/56.htm) * [프로그레스바](http://jacking75.cafe24.com/WTL/57.htm) * [핫키](http://jacking75.cafe24.com/WTL/58.htm) * [애니메이션](http://jacking75.cafe24.com/WTL/59.htm) * [드래그 리스트 박스](http://jacking75.cafe24.com/WTL/60.htm) * [확장 콤보 박스](http://jacking75.cafe24.com/WTL/61.htm) * [월간 달력](http://jacking75.cafe24.com/WTL/62.htm) * [일시 지정](http://jacking75.cafe24.com/WTL/63.htm) * [IP어드레스](http://jacking75.cafe24.com/WTL/64.htm) * [Pager](http://jacking75.cafe24.com/WTL/65.htm) | |

|  |  |
| --- | --- |
|  | 컨트룰러 확장 |
| * [슈퍼 클래스 화](http://jacking75.cafe24.com/WTL/66.htm) * [서브 클래스 화](http://jacking75.cafe24.com/WTL/67.htm) * [컨테인 윈도우](http://jacking75.cafe24.com/WTL/68.htm) * [메세지 리플렉션](http://jacking75.cafe24.com/WTL/69.htm) * [DDX/DDV](http://jacking75.cafe24.com/WTL/70.htm) * [오너 드로우](http://jacking75.cafe24.com/WTL/71.htm) * [커스텀 드로우](http://jacking75.cafe24.com/WTL/72.htm) | |

|  |  |
| --- | --- |
|  | 확장 컨트룰러 |
| * [비트맵 버튼](http://jacking75.cafe24.com/WTL/73.htm) * [체크 리스트 뷰](http://jacking75.cafe24.com/WTL/74.htm) * [하이퍼 링크](http://jacking75.cafe24.com/WTL/75.htm) * [모래 시계 커서](http://jacking75.cafe24.com/WTL/76.htm) | |

|  |  |
| --- | --- |
|  | 바 |
| * [툴바](http://jacking75.cafe24.com/WTL/77.htm) * [리바](http://jacking75.cafe24.com/WTL/78.htm) * [스테터스 바](http://jacking75.cafe24.com/WTL/79.htm) * [커맨드 바](http://jacking75.cafe24.com/WTL/80.htm) | |

|  |  |
| --- | --- |
|  | 공용 다이얼로그 |
| * [파일 선택 다이얼로그](http://jacking75.cafe24.com/WTL/81.htm) * [폴더 선택 다이얼로그](http://jacking75.cafe24.com/WTL/82.htm) * [폰트 선택 다이얼로그](http://jacking75.cafe24.com/WTL/83.htm) * [색 선택 다이얼로그](http://jacking75.cafe24.com/WTL/84.htm) * [검색 치환 선택 다이얼로그](http://jacking75.cafe24.com/WTL/85.htm) | |

|  |  |
| --- | --- |
|  | 프로퍼티 쉬트 |
| * [프로퍼티 쉬트](http://jacking75.cafe24.com/WTL/86.htm) * [위자드 모드](http://jacking75.cafe24.com/WTL/87.htm) | |

|  |  |
| --- | --- |
|  | 인쇄 |
| * [프린터 디바이스 컨텍스트](http://jacking75.cafe24.com/WTL/88.htm) * [인쇄 다이얼로그](http://jacking75.cafe24.com/WTL/89.htm) * [인쇄 프로퍼티 쉬트](http://jacking75.cafe24.com/WTL/90.htm) * [페이지 설정 다이얼로그](http://jacking75.cafe24.com/WTL/91.htm) * [인쇄 JOB](http://jacking75.cafe24.com/WTL/92.htm) * [인쇄 미리보기](http://jacking75.cafe24.com/WTL/93.htm) | |

|  |  |
| --- | --- |
|  | 뷰 |
| * [스크롤](http://jacking75.cafe24.com/WTL/94.htm) * [리스트 박스](http://jacking75.cafe24.com/WTL/95.htm) * [리스트 뷰](http://jacking75.cafe24.com/WTL/96.htm) * [트리 뷰](http://jacking75.cafe24.com/WTL/97.htm) * [탭](http://jacking75.cafe24.com/WTL/98.htm) * [에디터](http://jacking75.cafe24.com/WTL/99.htm) * [리치에디터](http://jacking75.cafe24.com/WTL/100.htm) * [폼](http://jacking75.cafe24.com/WTL/101.htm) | |

|  |  |
| --- | --- |
|  | 다이얼로그 사이즈 |
| * [다이얼로그](http://jacking75.cafe24.com/WTL/102.htm) * [폼](http://jacking75.cafe24.com/WTL/103.htm) * [윈도우](http://jacking75.cafe24.com/WTL/104.htm) | |

|  |  |
| --- | --- |
|  | MDI |
| * [MDI 윈도우](http://jacking75.cafe24.com/WTL/105.htm) * [MDI 커맨드 바](http://jacking75.cafe24.com/WTL/106.htm) | |

|  |  |
| --- | --- |
|  | 스플릿터 윈도우 |
| * [스플릿터 윈도우](http://jacking75.cafe24.com/WTL/107.htm) * [3페인 스플릿터 윈도우](http://jacking75.cafe24.com/WTL/108.htm) * [페인 컨테이너](http://jacking75.cafe24.com/WTL/109.htm) | |

|  |  |
| --- | --- |
|  | MRU 파일 리스트 |
| * [MRU 파일 리스트](http://jacking75.cafe24.com/WTL/110.htm) | |

|  |  |
| --- | --- |
|  | 부록 |
| * [WTL메세지 매크로 일람](http://jacking75.cafe24.com/WTL/111.htm) * 참고문헌 | |

▣ 시작

|  |
| --- |
| 본문서 「ATL/WTL」은 ,Microsoft이 제공하고 있는 템플릿 라이브러리인ATL(Active Template Library) 와 open source의 템플릿 라이브러리인WTL(Windows Template Library) 를 사용한Windows프로그래밍에 대한 문서입니다. 이하에ATLWTL의 개요를 나타냅니다.   * ATL 이라는 것은 ATL(Active Template Library) 이란 VisualC++에 부속되는 템플릿 라이브러리입니다. 주로COM를 서포트하는 라이브러리로서 알려져 있습니다만 Windows프로그래밍을 간략화하는 클래스도 갖추고 있습니다. 그러한 클래스를 사용하는 경우는 반드시COM의 지식은 필요 없습니다.   WTL 이라는 것은 WTL(Windows Template Library) 이란 주로Windows의 GUI부분( 컨트롤 , common dialog , 코맨드 바 , 페인컨테이너등 ) 을 서포트 하는 템플릿 라이브러리에서 ATL의 확장 라이브러리라고 말할 수 있습니다. 원래Microsoft가 무료로 제공(다만 정식적 서포트는 없음)하고 있었습니다만 , 2004연5월에 open source화 되었습니다. ATL(와)과 같이 ,Win32API를 랩퍼 클래스로 감싼 것으로 , Windows프로그래밍을 간략화 합니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | ATL/WTL프로그래밍의 특징 |
| ATL및WTL은 그 이름과 같이 C++템플릿을 베이스로 하고 있어 오버헤드가 매우 낮다고 하는 특징이 있습니다. 그리고 MFC와 같은 공유DLL를 별도 필요로 하지 않습니다. 아래에 , 윈도우 중앙에 「Hello, World」이라고 표시하기만 하면 되는 간단한 프로그램의 실행 파일 사이즈를 비교한 표를 나타냅니다. 덧붙여 모두Win32 Application프로젝트로 작성해 MFC의 경우만MFC스태틱 라이브러리를 사용하도록 설정을 변경합니다. 나머지의 설정은 디폴트인 채입니다.   |  |  |  | | --- | --- | --- | | SDK | ATL | MFC | | 36KB | 36KB | 88KB |   　ATL/WTL프로그래밍에서는MFC과 달라 ,WinMain()를 은폐 하지 않고 Document/View아키텍쳐 이기도 하지 않습니다. 그러나 ATL/WTL그리고 사용되는 클래스의 멤버 함수명 등은 MFC의 멤버 함수명과 같은 것이 많이 있습니다. 그러므로 ,SDK스타일의Windows프로그래밍과MFC의 양쪽 모두의 경험이 있으면(자) 보다 이해하기 쉬울 것입니다. 덧붙여 본문서는 그러한 기본적인 지식이 있는 것을 전제로 하고 있습니다.   　마지막으로 , 본문서의 작성은ATL/WTL프로그래밍의 학습과 동시 진행이기 때문에 갱신은 부정기에 행해지는 것을 양해 바랍니다. 그리고 할 수 있는 한 정보의 정확성에 주위를 기울일 생각입니다만 , 반드시 그 내용을 보증하는 것이 아닙니다. | |

▣ WTL의 인스톨

WTL(은)는 아래의 웹 주소에서 다운로드 할 수 있습니다.   
[Windows Template Library (WTL) 7.1](http://j2k.naver.com/j2k_loading.php/korean/www.microsoft.com/downloads/details.aspx?familyid=1BE1EB52-AA96-4685-99A5-4256737781C5)  
  
다운로드 후 파일을 해제 합니다. 풀린 파일은 다음과 같은 구성이 되어 있습니다.

AppWiz60/

Visual C++ 6.0 용무의 것 ATL/WTL AppWizard 이 들어간 폴더

AppWiz70/

Visual C++ 7.0/7.1(Visual C++ .NET 2002/2003) 용의 것 ATL/WTL AppWizard 이 들어간 폴더

AppWizCE/

eMbedded Visual C++ 3.0/4.0 용의 것 ATL/WTL AppWizard 이 들어간 폴더

include/

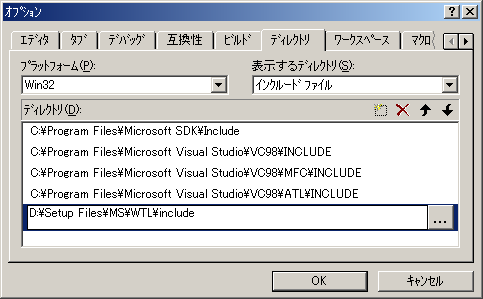
WTL를 구성하는 헤더 파일이 들어간 폴더

Samples/

샘플이 들어간 폴더

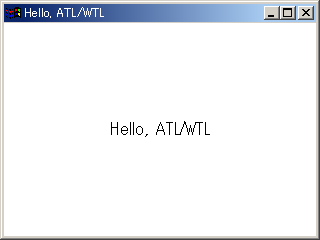
readme.htm

WTL에 대한 문서

Visual C++의 IDE에 ATL/WTL 용의 AppWizard을 추가하기 위해서는 AppWiz\*\*(\*\*는VisualC++의 버젼을 나타내는 숫자) 폴더에 들어가 있는 setup.js을 실행합니다. 본문서에서는Visual C++6.0을 사용하기 위해 AppWiz60폴더에 들어가있는setup.js을 실행합니다. setup.js를 실행하면 AppWizard파일이 VisualC++의 기정의 폴더에 카피되어 프로젝트의 신규 작성시에 ATL/WTL AppWizard 의 아이콘이 표시되게 됩니다.   
  
  
  
　다음에 WTL를 프로그램으로 사용하기 위해서 방금 전 압축을 푼 include폴더의 패스를 Visual C++의 인클루드 파일 패스에 추가합니다. Visual C++6.0그럼 , 메뉴의 [툴] - [옵션] 의 [디렉토리] 탭의 [표시하는 디렉토리] 에서 [인클루드 파일] 을 선택해 , 디렉토리 리스트에 WTL의 include 폴더 패스를 추가합니다.   
  
  
WTL의 인스톨 방법(영어)은 readme.htm 에도 기술되고 있습니다.

▣ 윈도우

우선 처음에 WTL를 사용하지 않고ATL만 을 사용해 템플릿 라이브러리에 의한 Windows프로그래밍의 기본을 해보고 싶다고 생각합니다.   
  
　ATL에 있어서의 가장 기본적인 윈도우 관련 클래스는CWindow입니다. CWindow클래스는 윈도우 핸들을 캡슐화해 윈도우에 관한 Win32API을 랩퍼 하는 많은 멤버 함수를 가집니다. 이것은 정확히MFC의 CWnd 클래스와 닮아 있습니다.   
  
　CWindow그리고 윈도우를 작성하려면 Create멤버 함수를 사용하거나 기존의 윈도우 핸들을 Attach멤버 함수로 Attach 할 필요가 있습니다.   
  
　그런데 CWindow클래스는 메세지로의 반응을 정의 할 수 없습니다. 여기서 만약MFC이라면CWnd의 상속 클래스를 만들어 예를 들면WM\_CREATE메세지로의 반응을 정의 하기 위해서는OnCreate을 오버라이드(override) 하지만 ATL은 CWindow의 상속 클래스인CWindowImpl클래스로부터 상속 클래스를 만들어 , 그 클래스내에서 메세지로의 응답을 정의합니다.   
  
　밑의 것은 윈도우 중앙에 「Hello, ATL/WTL」이라고 표시하기만 하면 되는 간단한 프로그램의 원시 코드입니다. 이 프로그램에서는 CWindowImpl클래스로부터 상속 클래스CMyWindow(이것이 메인 윈도우가 됩니다 ) 를 만들어 CMyWindow 클래스 내에서 WM\_PAINT와 WM\_DESTROY 메세지로의 응답을 정의하고 있습니다.   
덧붙여 이 프로그램은Win32Application프로젝트로 빌드 합니다.



|  |
| --- |
| // stdafx.h내  #include <atlbase.h>  extern CComModule \_Module;  #include <atlwin.h> |

|  |
| --- |
| // MainWindow.h내  // CWindowImpl의 상속 클래스로 메세지에의 반응을 정의  class CMyWindow : public CWindowImpl<CMyWindow>  {  public:  // 윈도우 클래스명을 등록  DECLARE\_WND\_CLASS(\_T("Hello"));  private:  // 메세지 맵  BEGIN\_MSG\_MAP(CMyWindow)  MESSAGE\_HANDLER(WM\_PAINT, OnPaint)  MESSAGE\_HANDLER(WM\_DESTROY, OnDestroy)  END\_MSG\_MAP()  LRESULT OnPaint(UINT, WPARAM, LPARAM, BOOL&){  PAINTSTRUCT ps;  HDC hDC = BeginPaint(&ps);  RECT rect;  GetClientRect(&rect);  DrawText(hDC, \_T("Hello, ATL/WTL"),  -1, &rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  EndPaint(&ps);  return 0;  }  LRESULT OnDestroy(UINT, WPARAM, LPARAM, BOOL&){  PostQuitMessage(0);  return 0;  }  }; |

|  |
| --- |
| // hello.cpp내  #include "stdafx.h"  #include "MainWindow.h"  CComModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.Create(NULL, CWindow::rcDefault,  \_T("Hello, ATL/WTL"), WS\_OVERLAPPEDWINDOW | WS\_VISIBLE);  MSG msg;  while(GetMessage(&msg, NULL, 0, 0) > 0){  TranslateMessage(&msg);  DispatchMessage(&msg);  }  \_Module.Term();  return msg.wParam;  } |

　우선 ,ATL을 사용하기 위한 헤더를 인클루드 합니다만 ,CComModule클래스의 인스턴스인 \_Module은ATL의 각 헤더로부터 참조되므로 글로벌하게 선언해 둡니다. 무엇보다 ,ATL7.0이후를 사용하고 있는 경우는 이 글로벌 선언은 필요없는 것 같습니다. 이\_Module는 ,\_tWinMain()의 최초와 최후로 초기화와 뒤처리를 하고 있습니다.   
  
　CMyWindow클래스는CWindowImpl클래스로부터 파생하고 있습니다만 ,CWindowImpl의 제1템플릿 인수에도 CMyWindow라는 이름 을 건네줍니다 ( 제2, 3템플릿 인수는 생략 가능합니다. 여기에서는 생략 하고 있습니다. ). CMyWindow클래스내에서는 , 윈도우 클래스명을DECLARE\_WND\_CLASS매크로에 의해 등록해 , 메세지 맵에 의해 메세지와 그에 대한 핸들러를 묶고 있습니다. 이번 예에서는 , 핸들러내의 처리는SDK스타일과 거의 동등합니다.

▣ 윈도우 클래스 정보

|  |
| --- |
| SDK스타일의Windows프로그래밍에서는 윈도우 클래스명이나 배경색 커서 등의 속성을 지정하기 위해서 WNDCLASSEX구조체를 정의하지만 ATL에는 그 작업을 단순화 하는 매크로나 구조체가 준비되어 있습니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 윈도우 클래스명 매크로 |
| 윈도우 클래스명을 지정하기 위한 매크로를 아래에 나타냅니다.   * DECLARE\_WND\_CLASS(윈도우 클래스명) 윈도우 클래스명만을 정의합니다. * DECLARE\_WND\_CLASS\_EX(윈도우 클래스명, 스타일, 배경색)  윈도우 클래스명 , 스타일 , 배경색을 정의합니다.   이러한 매크로는 public 선언으로 사용하지 않으면 안됩니다. 이러한 매크로를 사용해 윈도우 클래스명을 명시적으로 등록하지 않으면 ATL은 "ATL:00406060" 과 같은 이름을 자동적으로 등록합니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | CWndClassInfo구조체 |
| CWndClassInfo구조체는 윈도우 클래스명 매크로보다 윈도우의 속성을 세세하게 지정할 수 있습니다. CWndClassInfo(을)를 사용해 새로운 속성을 지정하기 위해서는 , CWindowImpl::GetWndClassInfo()을 오버라이드(override) 해 커스터마이즈 한 정적인CWndClassInfo인스턴스의 참조를 돌려줍니다.   |  | | --- | | class CMyWindow: public CWindowImpl<CMyWindow>  {  public:  static CWndClassInfo& GetWndClassInfo()  {  static CWndClassInfo wc =  {  {sizeof(WNDCLASSEX), CS\_HREDRAW | CS\_VREDRAW, StartWindowProc,  0, 0, NULL, NULL, NULL, (HBRUSH)(COLOR\_WINDOW + 1), NULL,  "MyWindow", NULL}, // WNDCLASSEX구조체  NULL, // 기존의 윈도우 클래스명  NULL, // 기존의 윈도우 프로시저  MAKEINTRESOURCE(IDC\_CURSOR1), // 커서 자원명  FALSE, // 시스템 커서라면TRUE, 그 이외는FALSE  0, // 등록이 끝난 윈도우 클래스의 식별자  \_T("") // ATL가 자동 생성한 윈도우 클래스명  };  return wc;  }  ...  }; | | |

▣ 윈도우 특성

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ATL그럼 , 윈도우의 스타일은 ,CWinTraits클래스에 의한 「윈도우 특성」으로서 표현할 수 있습니다. CWinTraits클래스는 윈도우 스타일과 확장 윈도우 스타일을 서포트합니다.   　지금까지의 예에서는 윈도우 작성시에CWindowImpl::Create()의 제4인수로 윈도우 스타일을 지정했습니다. 이 제4인수는 생략 가능합니다. 생략 했을 경우는 , 다음과 같이 선언되고 있는CControlWinTraits클래스가 디폴트 값으로서 사용됩니다.   |  | | --- | | typedef CWinTraits<WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS, 0>  CControlWinTraits; |   이것은 디폴트로는 자식과 형제를 클립 하는 가시 상태의 자식 윈도우를 작성하는 것을 의미합니다. CControlWinTraits클래스는 ,CWindowImpl클래스의 정의로 디폴트의 템플릿 인수로서 다음과 같이 사용되고 있습니다.   |  | | --- | | template <class T,  class TBase = CWindow,  class TWinTraits = CControlWinTraits>  class CWindowImpl : public ... |   이 때문에 CWindowImpl클래스의 제3 템플릿 인수에 독자적인 윈도우 특성을 지정하는 것에 의해 독자적인 윈도우 스타일을 지정하는 일도 가능합니다.   |  | | --- | | // 디폴트의 윈도우 특성을 가지는 윈도우( 제2, 3템플릿 인수는 생략 가능)  class CDefaultWindow : public CWindowImpl<CDefaultWindow, CWindow, CControlWinTraits>  {...};  // 독자적인 윈도우 특성을 가지는 윈도우  typedef CWinTraits<WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, WS\_EX\_CLIENTEDGE> CMyTraits;  class CMyWindow : public CWindowImpl<CMyWindow, CWindow, CMyTraits>  {...}; |   덧붙여 이러한 템플릿 인수에 의한 윈도우 특성은 CWindowImpl::Create()의 인수에 의해 오버라이드(override) 할 수 있습니다.   |  | | --- | | // 독자적인 윈도우 특성  typedef CWinTraits<WS\_OVERLAPPEDWINDOW | WS\_VISIBLE, 0> CMyTraits;  class CMyWindow : public CWindowImpl<CMyWindow, CWindow, CMyTraits>  {...};  // Create의 제4인수를 생략 하고 있으므로CMyTraits가 적용되는  // 즉 윈도우는WS\_OVERLAPPEDWINDOW과WS\_VISIBLE스타일을 가지는  CMyWindow wnd1;  wnd1.Create(NULL, CWindow::rcDefault, \_T("Hello"));  // Create의 제4인수를 지정해 있으므로 특성은 오버라이드(override) 되는  // 즉 윈도우는WS\_OVERLAPPEDWINDOW스타일만을 가진다  CMyWindow wnd2;  wnd2.Create(NULL, CWindow::rcDefault, \_T("Hello"), WS\_OVERLAPPEDWINDOW); | |

**▣ ATL의 메시지 맵**

|  |  |
| --- | --- |
| ATL/WTL프로그래밍에서는 메세지와 그에 대한 핸들러 함수를 메세지 맵에 의해 연결시킵니다.   |  | | --- | | class CMyWindow : public CWindowImpl<CMyWindow>  {  // 메세지 맵  BEGIN\_MSG\_MAP(CMyWindow)  MESSAGE\_HANDLER(WM\_PAINT, OnPaint)  MESSAGE\_HANDLER(WM\_DESTROY, OnDestroy)  END\_MSG\_MAP()  LRESULT OnPaint(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled){  // WM\_PAINT메세지에의 응답 처리  }  LRESULT OnDestroy(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled){  // WM\_DESTROY메세지에의 응답 처리  }  }; |   　윈도우가 메세지를 받아들이면 메세지 맵의 선두로부터 순서에 검색되므로 빈번하게 사용되는 메세지를 최초에 기술해 두면 좋을 것입니다. 메세지 맵 중에 대응하는 메세지 매크로가 발견되지 않으면 메세지는 디폴트 윈도우 프로시저에게 건네집니다.   　ATL에서 준비하는 메세지 매크로는 크게 나누어 3종류 있습니다. 그것은 메세지 핸들러 매크로, 커멘드 핸들러 매크로 , 통지 핸들러 매크로의 3개입니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 메세지 핸들러 매크로 |
| 메세지 핸들러 매크로는 모든 메세지를 대상으로 합니다. 메세지 핸들러 매크로에는 다음의 2개가 있습니다.  **MESSAGE\_HANDLER(메세지명, 핸들러명)** 지정된 메세지를 핸들러 함수에 맵 합니다.  **MESSAGE\_RANGE\_HANDLER(개시 위치의 메세지명, 종료 위치의 메세지명, 핸들러명)** 연속한 범위의 복수의 메세지를 핸들러에 맵 합니다.  메세지 핸들러 함수의 prototype를 이하에 나타냅니다.   |  | | --- | | LRESULT MessageHandler(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled); |   uMsg는 메세지를 식별해 wParam과 lParam은 메세지 파라미터입니다. 메세지 파라미터의 내용은 메세지의 종류에 의해 바뀝니다. bHandled은 메세지의 처리를 끝냈는지 어떠했는지를 나타내는 플래그입니다. bHandled하지만 핸들러 함수 중에서FALSE로 설정되어 있으면 메세지 맵의 나머지의 부분에서 그 메세지를 위한 핸들러가 별로 없는지 어떤지가 검색됩니다. bHandled은 핸들러 함수의 호출전에TRUE로 설정되므로 핸들러 함수가bHandled를 명시적으로FALSE로 설정하지 않는 한 그 이상의 핸들러 검색 처리는 행해지지 않습니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 커멘드 핸들러 매크로 |
| 커멘드 핸들러 매크로는 커멘드 메세지(WM\_COMMAND) 를 대상으로 합니다. 커멘드 핸들러 매크로에는 다음의 4개가 있습니다.  **COMMAND\_HANDLER(ID, 통지 코드, 핸들러명)** 지정된 컨트롤로부터의 지정된 통지 코드를 가지는 커멘드 메세지를 핸들러 함수에 맵 합니다.  **COMMAND\_ID\_HANDLER(ID, 핸들러명)**  지정된 컨트롤로부터의 임의의 통지 코드를 가지는 커멘드 메세지를 핸들러 함수에 맵 합니다.  **COMMAND\_CODE\_HANDLER(통지 코드, 핸들러명)** 임의의 컨트롤로부터의 지정된 통지 코드를 가지는 커멘드 메세지를 핸들러 함수에 맵 합니다.  **COMMAND\_RANGE\_HANDLER(개시 위치의 메세지명, 종료 위치의 메세지명, 핸들러명)** 연속한 범위의 컨트롤로부터의 커멘드 메세지를 핸들러 함수에 맵 합니다.  커멘드 핸들러 함수의 prototype를 이하에 나타냅니다.   |  | | --- | | LRESULT CommandHandler(WORD wNotifyCode, WORD wID, HWND hWndCtl, BOOL& bHandled); |   wNotifyCode 는 통지 코드 wID는 커멘드를 송신하고 있는 컨트롤의 식별자 hWndCtl는 커멘드를 송신하고 있는 컨트롤의 핸들 bHandled은 전술의 플래그입니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 통지 핸들러 매크로 |
| 통지 핸들러 매크로는 통지 메세지(WM\_NOTIFY) 를 대상으로 합니다. 통지 핸들러 매크로에는 다음의 4개가 있습니다.  **NOTIFY\_HANDLER(ID, 통지 코드, 핸들러명)** 지정된 컨트롤로부터의 지정된 통지 코드를 가지는 통지 메세지를 핸들러 함수에 맵 합니다.  **NOTIFY\_ID\_HANDLER(ID, 핸들러명)**  지정된 컨트롤로부터의 임의의 통지 코드를 가지는 통지 메세지를 핸들러 함수에 맵 합니다.  **NOTIFY\_CODE\_HANDLER(통지 코드, 핸들러명)** 임의의 컨트롤로부터의 지정된 통지 코드를 가지는 통지 메세지를 핸들러 함수에 맵 합니다.  **NOTIFY\_RANGE\_HANDLER(개시 위치의 메세지명, 종료 위치의 메세지명, 핸들러명)** 연속한 범위의 컨트롤로부터의 통지 메세지를 핸들러 함수에 맵 합니다.  통지 핸들러 함수의 prototype를 이하에 나타냅니다.   |  | | --- | | LRESULT NotifyHandler(int idCtrl, LPNMHDR pnmh, BOOL& bHandled); |   idCtrl은 통지를 송신하고 있는 컨트롤의 식별자 pnmh는 NMHDR구조체에의 포인터 bHandled는 전술의 플래그입니다. | |

**▣ WTL의 메시지 맵**

|  |  |
| --- | --- |
| 전회는ATL에 의해 제공되는 메세지 매크로에 대해 설명했습니다만 WTL도 메세지 매크로를 준비하고 있어 atlcrack.h 헤더로 정의되고 있습니다.   ATL3.0그리고WTL의 메세지 매크로를 사용하려면 , 메세지 맵을 BEGIN\_MSG\_MAP로부터 BEGIN\_MSG\_MAP\_EX로 변경할 필요가 있습니다.   ATL7.0/7.1그리고WTL의 메세지 매크로를 사용하는 경우는 CWindowImpl/CDialogImpl클래스의 상속 클래스에서는BEGIN\_MSG\_MAP인 채 로 좋습니다만 CWindowImpl/CDialogImpl클래스의 상속 클래스가 아닌 경우는 BEGIN\_MSG\_MAP\_EX으로 변경할 필요가 있습니다.   |  | | --- | | class CMyWindow : public CWindowImpl<CMyWindow>  {  // WTL의 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_DESTROY(OnDestroy)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  // WM\_PAINT메세지에의 응답 처리  }  void OnDestroy(){  // WM\_DESTROY메세지에의 응답 처리  }  }; |   이와 같이WTL에는 각 메세지 전용의 메세지 매크로가 있어 메세지 핸들러 함수에는 불필요한 인수가 없습니다.  WTL에는 ,ATL이 준비하는 메세지 핸들러 매크로 커멘드 핸들러 매크로 통지 핸들러 매크로의 확장판이 준비되어 있습니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 메세지 핸들러 매크로 |
| 메세지 핸들러 매크로는 모든 메세지를 대상으로 합니다. 메세지 핸들러 매크로에는 다음의 2개가 있습니다.  **MESSAGE\_HANDLER\_EX(메세지명, 핸들러명)** 지정된 메세지를 핸들러 함수에 맵 합니다.  **MESSAGE\_RANGE\_HANDLER\_EX(개시 위치의 메세지명, 종료 위치의 메세지명, 핸들러명)** 연속한 범위의 복수의 메세지를 핸들러에 맵 합니다.  메세지 핸들러 함수의 prototype를 이하에 나타냅니다.   |  | | --- | | LRESULT MessageHandler(UINT uMsg, WPARAM wParam, LPARAM lParam); |   uMsg는 메세지를 식별해 wParam과 lParam는 메세지 파라미터입니다. 메세지 파라미터의 내용은 메세지의 종류에 의해 바뀝니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 커멘드 핸들러 매크로 |
| 커멘드 핸들러 매크로는 커멘드 메세지(WM\_COMMAND) 를 대상으로 합니다. 커멘드 핸들러 매크로에는 다음의 5개가 있습니다.  COMMAND\_HANDLER\_EX(ID, 통지 코드, 핸들러명)  지정된 컨트롤로부터의 지정된 통지 코드를 가지는 커멘드 메세지를 핸들러 함수에 맵 합니다.  COMMAND\_ID\_HANDLER\_EX(ID, 핸들러명)  지정된 컨트롤로부터의 임의의 통지 코드를 가지는 커멘드 메세지를 핸들러 함수에 맵 합니다.  COMMAND\_CODE\_HANDLER\_EX(통지 코드, 핸들러명)  임의의 컨트롤로부터의 지정된 통지 코드를 가지는 커멘드 메세지를 핸들러 함수에 맵 합니다.  COMMAND\_RANGE\_HANDLER\_EX(개시 위치의 메세지명, 종료 위치의 메세지명, 핸들러명)  연속한 범위의 컨트롤로부터의 커멘드 메세지를 핸들러 함수에 맵 합니다.  COMMAND\_RANGE\_CODE\_HANDLER\_EX(개시 위치의 메세지명, 종료 위치의 메세지명, 통지 코드, 핸들러명)  연속한 범위의 컨트롤로부터의 지정된 통지 코드를 가지는 커멘드 메세지를 핸들러 함수에 맵 합니다.  커멘드 핸들러 함수의 prototype를 이하에 나타냅니다.   |  | | --- | | void CommandHandler(UINT uNotifyCode, int nID, HWND hWndCtl); |   uNotifyCode는 통지 코드 nID는 커멘드를 송신하고 있는 컨트롤의 식별자 hWndCtl는 커멘드를 송신하고 있는 컨트롤의 핸들입니다. | |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 통지 핸들러 매크로 |
| 통지 핸들러 매크로는 통지 메세지(WM\_NOTIFY) 를 대상으로 합니다. 통지 핸들러 매크로에는 다음의 5개가 있습니다.   * NOTIFY\_HANDLER\_EX(ID, 통지 코드, 핸들러명)  지정된 컨트롤로부터의 지정된 통지 코드를 가지는 통지 메세지를 핸들러 함수에 맵 합니다. * NOTIFY\_ID\_HANDLER\_EX(ID, 핸들러명)  지정된 컨트롤로부터의 임의의 통지 코드를 가지는 통지 메세지를 핸들러 함수에 맵 합니다. * NOTIFY\_CODE\_HANDLER\_EX(통지 코드, 핸들러명)  임의의 컨트롤로부터의 지정된 통지 코드를 가지는 통지 메세지를 핸들러 함수에 맵 합니다. * NOTIFY\_RANGE\_HANDLER\_EX(개시 위치의 메세지명, 종료 위치의 메세지명, 핸들러명)  연속한 범위의 컨트롤로부터의 통지 메세지를 핸들러 함수에 맵 합니다. * NOTIFY\_RANGE\_CODE\_HANDLER\_EX(개시 위치의 메세지명, 종료 위치의 메세지명, 통지 코드, 핸들러명)  연속한 범위의 컨트롤로부터의 지정된 통지 코드를 가지는 통지 메세지를 핸들러 함수에 맵 합니다.   통지 핸들러 함수의 prototype를 이하에 나타냅니다.   |  | | --- | | LRESULT NotifyHandler(LPNMHDR pnmh); |   pnmh는NMHDR구조체에의 포인터입니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 메세지 처리의 종료 통지 |
| 그런데 ,ATL의 메세지 매크로를 사용하고 있는 경우는 핸들러 함수의 prototype에 처리를 끝냈는지 어떠했는지를 통지하는 플래그가 있었습니다. 다음에 나타내는 것은ATL판 메세지 핸들러 함수의 prototype입니다.   |  | | --- | | // bHandled하지만 처리를 끝냈는지 어떠했는지를 통지하는 플래그  LRESULT MessageHandler(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled); |   그러나 WTL의 메세지 핸들러 함수에는 그러한 플래그가 준비되어 있지 않습니다.   그런데 WTL에서 메세지 핸들러 함수내에서SetMsgHandled() 라고 하는 함수를 호출하는 것에 의해 메세지의 처리를 끝냈는지 어떠했는지를 통지합니다. SetMsgHandled()의 인수에 true를 건네주었을 경우는 메세지의 처리를 끝낸 것을 의미해 false를 건네주었을 경우는 메세지 맵 내에서 한층 더 이 메세지용의 핸들러가 있는지 없는지 검색하는 것을 의미합니다. WTL그럼 기본적으로 메세지 핸들러 함수의 호출전에 true가 설정되어 있기 때문에 메세지 맵을 검색시키고 싶은 경우만 명시적으로 false를 설정할 필요가 있습니다. | |

**▣ 체인**

|  |  |  |
| --- | --- | --- |
| ATL에는 체인이라고 하는 구조가 있습니다. 이것은 메세지 맵을 다른 메세지 맵과 연결하는 구조입니다. 왜 이러한 구조가 필요한가를 이하의 프로그램편을 예로 설명합니다.   |  | | --- | | // base class  class CBaseWindow : public CWindowImpl<CBaseWindow>  {  BEGIN\_MSG\_MAP(CBaseWindow)  MESSAGE\_HANDLER(WM\_DESTROY, OnDestroy)  END\_MSG\_MAP()  LRESULT OnDestroy(UINT, WPARAM, LPARAM, BOOL&)  {  PostQuitMessage(0);  return 0;  }  };  // 상속 클래스  class CMessageWindow : public CBaseWindow  {  BEGIN\_MSG\_MAP(CMessageWindow)  MESSAGE\_HANDLER(WM\_LBUTTONDOWN, OnButtonDown)  END\_MSG\_MAP()  LRESULT OnButtonDown(UINT, WPARAM, LPARAM, BOOL&)  {  MessageBox(\_T("Hello!"));  return 0;  }  }; |   　상기의 프로그램편은 , 윈도우를 종료하는 핸들러만을 가진 base class(CBaseWindow) 와 그 클래스로부터 파생한 클래스(CMessageWindow) 의 정의입니다. 상속 클래스에서는 왼쪽 mouse button를 눌렀을 때에 "Hello!" 라고 하는 메시지 박스를 표시하도록 하고 있습니다.   　여기서 상속 클래스인CMessageWindow클래스로부터 윈도우를 작성해 표시했다고 해도 이 윈도우는 종료할 수 없습니다. 즉 , 이 윈도우에WM\_DESTROY메세지가 보내져도 , 자동적으로 base class에는 보내지지 않고 , base class의 것OnDestroy()은 불려 가지 않습니다.   　이 문제를 해결하는 것이 체인이라고 하는 구조입니다. 즉 , 상속 클래스의 메세지 맵과 base class의 메세지 맵을 연결해 상속 클래스로 메세지 핸들러가 발견되지 않는 경우는 base class의 메세지 맵을 검색하도록 변경합니다. 이것은 기본 클래스에의 체인이라고 불립니다. 기본 클래스에의 체인을 실현하기 위해서는 상속 클래스의 메세지 맵내에서 CHAIN\_MSG\_MAP매크로를 사용합니다.   |  | | --- | | BEGIN\_MSG\_MAP(CMessageWindow)  MESSAGE\_HANDLER(WM\_LBUTTONDOWN, OnButtonDown)  CHAIN\_MSG\_MAP(CBaseWindow) // 체인처의 base class를 지정  END\_MSG\_MAP() |   　이와 같이 ATL그럼 상속 클래스에 해당하는 메세지 핸들러가 없을 때 메세지가 자동적으로 base class에 보내질 것은 없습니다. 이것은 ATL의 아키텍쳐가 다중 상속을 기반으로 하고 있기 때문입니다. 다중 상속에서는 base class가 복수 존재하기 위해 어느 base class의 메세지 맵에 메세지를 보낼까를 체인에 의해 명시적으로 지정하지 않으면 안 됩니다.   ATL(은)는CHAIN\_MSG\_MAP()을 포함해 다음과 같은 체인 매크로를 준비해 있습니다.   * CHAIN\_MSG\_MAP(클래스명)  지정된 클래스의 메세지 맵에 체인 합니다. * CHAIN\_MSG\_MAP\_MEMBER(인스턴스명) 지정된 인스턴스의 메세지 맵에 체인 합니다. |

**▣ 대체 메시지 맵**

전회 설명한 체인에 의해 base class의 메세지 맵은 복수의 상속 클래스로부터의 메세지를 처리할 수 있게 됩니다. 이하에 나타내는 것은 상속 클래스의 윈도우상에서 클릭하면, WM\_LBUTTONDOWN 메세지가 체인처의 base class에 보내져 메시지 박스가 표시된다고 하는 예입니다.

|  |
| --- |
| // base class  class CBaseMsgWindow : public CWindowImpl<CBaseMsgWindow>  {  BEGIN\_MSG\_MAP(CBaseMsgWindow)  MESSAGE\_HANDLER(WM\_LBUTTONDOWN, OnButtonDown)  END\_MSG\_MAP()  LRESULT OnButtonDown(UINT, WPARAM, LPARAM, BOOL&)  {  MessageBox(\_T("Hello!"));  return 0;  }  };  // 상속 클래스 1  class CHelloWindow : public CBaseMsgWindow  {  BEGIN\_MSG\_MAP(CHelloWindow)  ...  CHAIN\_MSG\_MAP(CBaseMsgWindow) // 체인처의 base class를 지정  END\_MSG\_MAP()  };  // 상속 클래스 2  class CHelloWindow2 : public CBaseMsgWindow  {  BEGIN\_MSG\_MAP(CHelloWindow2)  ...  CHAIN\_MSG\_MAP(CBaseMsgWindow) // 체인처의 base class를 지정  END\_MSG\_MAP()  }; |

이 예에서는 두 개의 상속 클래스와도 결과적으로 같은OnButtonDown()이라고 하는 메세지 핸들러를 호출합니다.   
  
　그런데 상기의 방법에서는 상속 클래스에 의해 다른 메세지 핸들러를 호출하고 싶다고 생각하고 있어도 당연합니다만 언제나 같은OnButtonDown()밖에 불려 가지 않습니다. 그러나 ATL의 「대체 메세지 맵」을 사용하면 상속 클래스에 의해 다른 메세지 핸들러를 호출하는 것이 가능합니다.   
  
다음에 나타내는 것은 base class의 메세지 맵을 세 개의 부분에 분할해 대체 메세지 맵을 작성해 상속 클래스로부터 base class의 대체 메세지 맵에 체인 한다고 하는 예입니다. 메세지 맵을 분할하려면 ,ALT\_MSG\_MAP매크로를 사용합니다.

|  |
| --- |
| // base class  class CBaseMsgWindow : public CWindowImpl<CBaseMsgWindow>  {  // 메세지 맵을 세 개의 부분에 분할  BEGIN\_MSG\_MAP(CBaseMsgWindow)  MESSAGE\_HANDLER(WM\_LBUTTONDOWN, OnButtonDown)  ALT\_MSG\_MAP(1)  MESSAGE\_HANDLER(WM\_LBUTTONDOWN, OnButtonDown2)  ALT\_MSG\_MAP(2)  MESSAGE\_HANDLER(WM\_LBUTTONDOWN, OnButtonDown3)  END\_MSG\_MAP()  LRESULT OnButtonDown(UINT, WPARAM, LPARAM, BOOL&)  {  MessageBox(\_T("Hello!"));  return 0;  }    LRESULT OnButtonDown2(UINT, WPARAM, LPARAM, BOOL&)  {  MessageBox(\_T("Hello, ATL!"));  return 0;  }    LRESULT OnButtonDown3(UINT, WPARAM, LPARAM, BOOL&)  {  MessageBox(\_T("Hello, ATL/WTL!"));  return 0;  }  };  // 상속 클래스 1  class CHelloWindow : public CBaseMsgWindow  {  BEGIN\_MSG\_MAP(CHelloWindow)  ...  CHAIN\_MSG\_MAP(CBaseMsgWindow) // 통상의 체인  END\_MSG\_MAP()  };  // 상속 클래스 2  class CHelloWindow2 : public CBaseMsgWindow  {  BEGIN\_MSG\_MAP(CHelloWindow2)  ...  CHAIN\_MSG\_MAP\_ALT(CBaseMsgWindow, 1) // 대체 메세지 맵에 체인  END\_MSG\_MAP()  }; |

　ALT\_MSG\_MAP매크로에는 대체 메세지 맵을 식별하는 번호를 지정합니다. 상기의 예에서는 전술대로 base class의 메세지 맵을 ALT\_MSG\_MAP매크로에 의해 세 개의 부분에 분할하고 있습니다. 첫째는 디폴트의 메세지 맵으로, 둘째는 번호 1 로 식별되는 대체 메세지 맵, 셋째는 번호 2 로 식별되는 대체 메세지 맵입니다. 덧붙여 디폴트의 메세지 맵에는 암묵중에 0 라고 하는 번호를 할당할 수 있습니다.

|  |
| --- |
| // 메세지 맵을 세 개의 부분에 분할  BEGIN\_MSG\_MAP(CBaseMsgWindow)  // 첫째  MESSAGE\_HANDLER(WM\_LBUTTONDOWN, OnButtonDown)  ALT\_MSG\_MAP(1)  // 둘째  MESSAGE\_HANDLER(WM\_LBUTTONDOWN, OnButtonDown2)  ALT\_MSG\_MAP(2)  // 셋째  MESSAGE\_HANDLER(WM\_LBUTTONDOWN, OnButtonDown3)  END\_MSG\_MAP() |

　이것으로 base class에 대체 메세지 맵을 준비할 수 있었으므로 다음에 상속 클래스로부터 대체 메세지 맵에 체인 하는 방법입니다만 그러기 위해서는 CHAIN\_MSG\_MAP\_ALT매크로를 사용합니다. CHAIN\_MSG\_MAP\_ALT매크로에는 체인처의 클래스명과 그 클래스에 있는 대체 메세지 맵의 번호를 지정합니다.   
  
상기의 예에서는 CHelloWindow클래스에서는 CBaseMsgWindow클래스에 통상의 체인을 하고 있을 뿐이므로 CBaseMsgWindow클래스의 디폴트의 메세지 맵에 있는 OnButtonDown()이 불려 갑니다. CHelloWindow2클래스에서는 CBaseMsgWindow클래스의 대체 메세지 맵 번호 1 에 체인 하고 있기 때문에 OnButtonDown2()가 불려 갑니다.   
  
ATL는CHAIN\_MSG\_MAP\_ALT()을 포함해 다음과 같은 대체 메세지 맵용 체인 매크로를 준비해 있습니다.

**CHAIN\_MSG\_MAP\_ALT(클래스명, 대체 메세지 맵 번호)**지정된 클래스의 대체 메세지 맵에 체인 합니다.

**CHAIN\_MSG\_MAP\_ALT\_MEMBER(인스턴스명, 대체 메세지 맵 번호)**지정된 인스턴스의 대체 메세지 맵에 체인 합니다.

**▣ 메시지 루프**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ATL만 을 사용해 작성한 「Hello, ATL/WTL」프로그램에서는 메시지 루프를SDK스타일로 썼습니다.   |  | | --- | | MSG msg;  while(GetMessage(&msg, NULL, 0, 0) > 0){  TranslateMessage(&msg);  DispatchMessage(&msg);  } |   WTL에는 메시지 루프를 캡슐화하기 위해서CMessageLoop라고 하는 클래스가 준비되어 있습니다. 여기에서는 이CMessageLoop클래스를 사용해 「Hello, ATL/WTL」프로그램 고쳐 씁니다. 덧붙여 이번으로부터WTL의 atlcrack.h헤더로 정의되고 있는 메세지 크래커를 사용합니다.   |  | | --- | | // stdafx.h안  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module; // CComModule으로부터CAppModule에 옮겨놓는  #include <atlwin.h>  #include <atlcrack.h> // WTL메세지 크래커를 사용하기 (위해)때문에 |  |  | | --- | | // MainWindow.h동안  class CMyWindow : public CWindowImpl<CMyWindow>  {  public:  // 윈도우 클래스명을 등록  DECLARE\_WND\_CLASS(\_T("Hello"));  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_DESTROY(OnDestroy)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  PAINTSTRUCT ps;  HDC hDC = BeginPaint(&ps);  RECT rect;  GetClientRect(&rect);  DrawText(hDC, \_T("Hello, ATL/WTL"),  -1, &rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  EndPaint(&ps);  }  void OnDestroy(){  PostQuitMessage(0);  }  }; |  |  | | --- | | // hello.cpp내  #include "stdafx.h"  #include "MainWindow.h"  CAppModule \_Module; // CComModule로부터CAppModule에 옮겨놓는  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.Create(NULL, CWindow::rcDefault,  \_T("Hello, ATL/WTL"), WS\_OVERLAPPEDWINDOW | WS\_VISIBLE);  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　우선 stdafx.h안에서는 CMessageLoop클래스를 사용하기 위해서atlapp.h헤더를 인클루드 합니다. 그리고 CMessageLoop클래스는CAppModule클래스에 의해 관리되기 (위해)때문에 지금까지 기술해 온CComModule클래스를CAppModule클래스에 옮겨놓습니다. CAppModule클래스도 atlapp.h 헤더에 선언되고 있기 때문에 CAppModule클래스를 사용하기 전에atlapp.h헤더를 인클루드 합니다. 한층 더WTL의 메세지 크래커를 사용하기 위해서 atlcrack.h헤더를 인클루드 하고 있습니다.   　CMyWindow클래스에서는 메세지와 메세지 핸들러를 묶기 위해서 WTL의 메세지 크래커를 사용하고 있습니다. 본문서에서는ATL3.0을 사용하고 있기)때문에 BEGIN\_MSG\_MAP를BEGIN\_MSG\_MAP\_EX에 옮겨놓고 있습니다.   　\_tWinMain()안에서는 ,CMessageLoop클래스의 인스턴스를 작성해 그것을CAppModule::AddMessageLoop()로 CAppModule내부의 맵에 추가하고 있습니다. 메시지 루프는CMessageLoop::Run()에 의해 실행되어 필요없게 된 메시지 루프는CAppModule::RemoveMessageLoop()에 의해 CAppModule내부의 맵으로부터 삭제합니다.   　이와 같이WTL에서는 ,CMessageLoop클래스에 의해 캡슐화된 메시지 루프를 CAppModule클래스에 의해 관리하고 있습니다. CAppModule클래스의 내부에서는 메시지 루프와 경향 threadID를 세트로 해 맵에 등록하고 있습니다. 즉 , 글로벌 선언된CAppModule클래스의 인스턴스인\_Module은 복수의 thread내에 있는 메시지 루프를 일원 관리하게 됩니다. |

**▣ 메시지 필터**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 메시지 루프에CMessageLoop클래스를 사용하면 CMessageFilter클래스를 사용해 메세지 필터를 추가할 수가 있습니다. 메세지 필터를 추가하면 CMessageLoop클래스내의 메시지 루프로 ::TranslateMessage()가 불려 가기 전에 무엇인가 처리를 시킬 수가 있습니다.   여기에서는 이CMessageFilter클래스를 사용해 전회의 메시지 루프를 변경한 「Hello, ATL/WTL」프로그램에 메세지 필터를 추가합니다. 변경하는 것은CMyWindow클래스만입니다.   |  | | --- | | // stdafx.h안  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module; // CComModule으로부터CAppModule에 옮겨놓는  #include <atlwin.h>  #include <atlcrack.h> // WTL메세지 크래커를 사용하기 (위해)때문에 |  |  | | --- | | // MainWindow.h동안  // base class에CMessageFilter를 추가  class CMyWindow : public CWindowImpl<CMyWindow>, public CMessageFilter  {  public:  // 윈도우 클래스명을 등록  DECLARE\_WND\_CLASS(\_T("Hello"));  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_DESTROY(OnDestroy)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  PAINTSTRUCT ps;  HDC hDC = BeginPaint(&ps);  RECT rect;  GetClientRect(&rect);  DrawText(hDC, \_T("Hello, ATL/WTL"),  -1, &rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  EndPaint(&ps);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  return 0;  }  void OnDestroy(){  PostQuitMessage(0);  }  }; |  |  | | --- | | // hello.cpp내  #include "stdafx.h"  #include "MainWindow.h"  CAppModule \_Module; // CComModule로부터CAppModule에 옮겨놓는  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.Create(NULL, CWindow::rcDefault,  \_T("Hello, ATL/WTL"), WS\_OVERLAPPEDWINDOW | WS\_VISIBLE);  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　CMessageFilter클래스는 메인 윈도우인CMyWindow클래스의 base class로서 사용하고 있습니다. CMessageFilter클래스에는PreTranslateMessage()이라고 하는 순수 가상 함수가 한 개 있을 뿐입니다.   |  | | --- | | // atlapp.h안  class CMessageFilter  {  public:  virtual BOOL PreTranslateMessage(MSG\* pMsg) = 0;  }; |   CMessageFilter클래스의 상속 클래스인CMyWindow클래스 에서는 이PreTranslateMessage()를 구현합니다. 이번 예에서는FALSE을 돌려주고 있을 뿐입니다만 이것은 굳이 하지 않고 통상의 처리를 계속한다고 하는 것을 의미합니다.   　CMyWindow클래스에서는WM\_CREATE메세지 핸들러를 추가해 거기서AddMessageFilter()를 호출하는 것에 의해 메시지 루프에 이 윈도우용의 메세지 필터를 추가하고 있습니다. 이것에 의해 CMessageLoop클래스내의 메시지 루프로 ::TranslateMessage()가 불려 가기 전에CMyWindow클래스에서 구현되었던 PreTranslateMessage()가 불려 가게 됩니다. |

**▣ 아이들 핸들러**

메시지 루프에CMessageLoop클래스를 사용하면 CIdleHandler클래스를 사용해 아이들 핸들러를 추가할 수가 있습니다. 아이들 핸들러를 추가하면 CMessageLoop클래스내의 메시지 루프로 처리하는 메세지가 없을 때에 아이들 처리를 시킬 수가 있습니다.   
  
여기에서는 이CIdleHandler클래스를 사용해 전회의 메세지 필터를 추가한 「Hello, ATL/WTL」프로그램에 아이들 핸들러를 추가합니다. 변경하는 것은CMyWindow클래스 만입니다.

|  |
| --- |
| // stdafx.h안  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module; // CComModule으로부터CAppModule에 옮겨놓는  #include <atlwin.h>  #include <atlcrack.h> // WTL메세지 크래커를 사용하기 (위해)때문에 |

|  |
| --- |
| // MainWindow.h동안  // base class에CMessageFilterCIdleHandler를 추가  class CMyWindow : public CWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler  {  public:  // 윈도우 클래스명을 등록  DECLARE\_WND\_CLASS(\_T("Hello"));  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 아이돌 처리  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_DESTROY(OnDestroy)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  PAINTSTRUCT ps;  HDC hDC = BeginPaint(&ps);  RECT rect;  GetClientRect(&rect);  DrawText(hDC, \_T("Hello, ATL/WTL"),  -1, &rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  EndPaint(&ps);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnDestroy(){  PostQuitMessage(0);  }  }; |

|  |
| --- |
| // hello.cpp내  #include "stdafx.h"  #include "MainWindow.h"  CAppModule \_Module; // CComModule로부터CAppModule에 옮겨놓는  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.Create(NULL, CWindow::rcDefault,  \_T("Hello, ATL/WTL"), WS\_OVERLAPPEDWINDOW | WS\_VISIBLE);  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |

　CIdleHandler클래스는 메인 윈도우인CMyWindow클래스의 base class로서 사용하고 있습니다. CIdleHandler클래스에는OnIdle()이라고 하는 순수 가상 함수가 한 개 있을 뿐입니다.

|  |
| --- |
| // atlapp.h안  class CIdleHandler  {  public:  virtual BOOL OnIdle() = 0;  }; |

CIdleHandler클래스의 상속 클래스인CMyWindow클래스 에서는 이OnIdle()를 구현합니다. 이번 예에서는FALSE을 돌려주고 있습니다만 이 값은 체크되어 있지 않은 것 같습니다.   
  
　그리고 CMyWindow클래스의 WM\_CREATE메세지 핸들러에서는 AddIdleHandler()을 호출하는 것에 의해 메시지 루프에 이 윈도우용의 아이들 핸들러를 추가하고 있습니다. 이것에 의해 아이들 시가 될 때마다 CMyWindow클래스에서 구현했던 OnIdle()가 불려 가게 됩니다.

**▣ GDI > 디바이스 컨텍스트**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 지금까지의 「Hello, ATL/WTL」프로그램에서는 WM\_PAINT메세지 핸들러의 처리를SDK스타일로 써 왔습니다.   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  PAINTSTRUCT ps;  HDC hDC = BeginPaint(&ps);  RECT rect;  GetClientRect(&rect);  DrawText(hDC, \_T("Hello, ATL/WTL"),  -1, &rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  EndPaint(&ps);  } |   　WTL은 GDI을 서포트 하는 클래스를 준비해 있습니다. 다음에 나타내는 것은 전회의 아이들 핸들러를 추가한 「Hello, ATL/WTL」프로그램의 WM\_PAINT메세지 핸들러를 WTL의GDI서포트 클래스를 사용해 고쳐 쓴 예입니다.   |  | | --- | | // stdafx.h안  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h> // CRect클래스를 사용하기 (위해)때문에 |  |  | | --- | | // MainWindow.h동안  class CMyWindow : public CWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler  {  public:  // 윈도우 클래스명을 등록  DECLARE\_WND\_CLASS(\_T("Hello"));  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 아이들 처리  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_DESTROY(OnDestroy)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnDestroy(){  PostQuitMessage(0);  }  }; |  |  | | --- | | // hello.cpp내  #include "stdafx.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.Create(NULL, CWindow::rcDefault,  \_T("Hello, ATL/WTL"), WS\_OVERLAPPEDWINDOW | WS\_VISIBLE);  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　GDI서포트 클래스는atlgdi.h헤더에 정의되고 있습니다만 이 헤더는atlapp.h헤더를 인클루드 한 시점에서 자동적으로 인클루드 되고 있습니다.   　이번은RECT구조체를 베이스로 한다. CRect라고 하는 클래스를 사용하기 위해서 stdafx.h안에서 atlmisc.h라고 하는 헤더를 인클루드 합니다. 이atlmisc.h에는CRect클래스의 외 SIZE구조체를 베이스로 하는CSize POINT구조체를 베이스로 하는CPoint 캐릭터 라인을 서포트하는CString등이 정의되고 있습니다. atlmisc.h그리고 정의되고 있는 이러한 클래스는MFC의 동명의 클래스와 거의 같은 기능을 갖추고 있습니다.   　다음에 ,WM\_PAINT메세지 핸들러를 고쳐 씁니다. 우선 ,CPaintDC클래스의 인스턴스를 작성합니다. 이 클래스는 constructor 으로::BeginPaint()를 호출해 디바이스 콘텍스트를 취득해 소멸자로::EndPaint()를 호출합니다. WM\_PAINT메세지 핸들러에서는 그 후 클라이언트 영역의 구형을 취득해 중앙에 「Hello, ATL/WTL」이라고 하는 캐릭터 라인을 표시하고 있습니다.   　이번은 디바이스 컨텍스트를 취득하기 위해서CPaintDC클래스를 사용했습니다만 WTL은CPaintDC클래스를 포함해 다음과 같은 디바이스 컨텍스트 취득 클래스를 준비해 있습니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | CPaintDC |
| WM\_PAINT메세지 핸들러 전용의 디바이스 콘텍스트 취득 클래스입니다. 전술대로 constructor　 　 으로 ::BeginPaint()를 호출해 디바이스 콘텍스트를 취득해 소멸자로 ::EndPaint()를 호출합니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | CClientDC |
| WM\_PAINT메세지 핸들러 이외의 장소에서 윈도우의 클라이언트 영역에 묘화하기 위한 디바이스 컨텍스트를 취득하는 클래스입니다. constructor 으로 ::GetDC()를 호출해 디바이스 컨텍스트를 취득해 소멸자로 ::ReleaseDC()를 호출해 디바이스 콘텍스트를 해방합니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | CWindowDC |
| 윈도우의 클라이언트 영역 뿐만이 아니라 비클라이언트 영역에도 묘화하기 위한 디바이스 콘텍스트를 취득하는 클래스입니다. constructor 으로 ::GetWindowDC()를 호출해 디바이스 컨텍스트를 취득해 소멸자로 ::ReleaseDC()를 호출해 디바이스 컨텍스트를 해방합니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | CEnhMetaFileDC |
| 확장 메타파일 디바이스 콘텍스트를 작성합니다.   이러한 클래스는 모두CDC라고 하는 클래스로부터 파생하고 있습니다. CDC클래스는 MFC의 동명의 클래스와 같이 디바이스 콘텍스트 핸들과 거기에 관한Win32API을 캡슐화합니다. WTL의CDC클래스는 게다가DIB나OpenGL도 서포트합니다. | |

▣ GDI > 펜

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WTL그럼 GDI오브젝트의 펜을CPenT이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는bool값으로 true의 경우는 소멸자로DeleteObject()를 호출해 false의 경우는 호출하지 않습니다. atlgdi.h헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlgdi.h내  typedef CPenT<false> CPenHandle;  typedef CPenT<true> CPen; |   　WTL의CPen클래스는 ,MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CPen클래스를 사용해 클라이언트 구형에 접하는 푸른 타원을 묘화 하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl04-02/01.png   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 굵기2로 푸른 실선의 펜을 작성해 선택  CPen pen;  pen.CreatePen(PS\_SOLID, 2, RGB(0, 0, 255));  HPEN hOldPen = dc.SelectPen(pen);  // 클라이언트 구형을 취득해 타원을 묘화  CRect rect;  GetClientRect(rect);  dc.Ellipse(rect);  // 원의 펜을 선택  dc.SelectPen(hOldPen);  } |   　이 예에서는 우선 펜 오브젝트를 작성해 그것을SelectPen()로 현재의 펜으로서 선택하고 있습니다. SelectPen()(은)는 인수에HPEN형태의 변수를 받습니다만 , CPenT클래스는HPEN변환 연산자를 가지고 있으므로 그대로 건네줄 수가 있습니다. SelectPen()은 이전의 펜 핸들을 돌려주므로 이것을hOldPen이라고 하는 변수에 보존해 둡니다.   다음에 원을 묘화 합니다. 굵기가 2 로 푸른 실선의 원이 묘화 됩니다.   마지막으로 방금전 보존해 둔 원의 펜 핸들에 되돌리기 위해 재차SelectPen()를 호출해 원의 펜 핸들을 선택합니다.   　그런데 WTL는 스톡 펜도 서포트하고 있습니다. 이하에 나타내는 것은 스톡 펜에 의해 검은 실선의 타원을 묘화 하는 예입니다.   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 스톡 펜을 선택  HPEN hOldPen = dc.SelectStockPen(BLACK\_PEN);  // 클라이언트 구형을 취득해 , 엔을 묘화  CRect rect;  GetClientRect(rect);  dc.Ellipse(rect);  // 원의 펜을 선택  dc.SelectPen(hOldPen);  } |   그리고 ,atlmisc.h헤더에 정의되고 있다AtlGetStockPen()라고 하는 글로벌 함수를 사용하면 스톡 펜의 핸들을 취득할 수가 있습니다.   |  | | --- | | // 스톡 펜의 핸들을 취득  HPEN hStockPen = AtlGetStockPen(BLACK\_PEN); | |

**▣ GDI > 브러쉬**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WTL그럼 GDI오브젝트의 펜을CPenT이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는bool값으로 true의 경우는 소멸자로DeleteObject()를 호출해 false의 경우는 호출하지 않습니다. atlgdi.h헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlgdi.h내  typedef CPenT<false> CPenHandle;  typedef CPenT<true> CPen; |   　WTL의CPen클래스는 ,MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CPen클래스를 사용해 클라이언트 구형에 접하는 푸른 타원을 묘화 하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl04-02/01.png   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 굵기2로 푸른 실선의 펜을 작성해 선택  CPen pen;  pen.CreatePen(PS\_SOLID, 2, RGB(0, 0, 255));  HPEN hOldPen = dc.SelectPen(pen);  // 클라이언트 구형을 취득해 타원을 묘화  CRect rect;  GetClientRect(rect);  dc.Ellipse(rect);  // 원의 펜을 선택  dc.SelectPen(hOldPen);  } |   　이 예에서는 우선 펜 오브젝트를 작성해 그것을SelectPen()로 현재의 펜으로서 선택하고 있습니다. SelectPen()(은)는 인수에HPEN형태의 변수를 받습니다만 , CPenT클래스는HPEN변환 연산자를 가지고 있으므로 그대로 건네줄 수가 있습니다. SelectPen()은 이전의 펜 핸들을 돌려주므로 이것을hOldPen이라고 하는 변수에 보존해 둡니다.   다음에 원을 묘화 합니다. 굵기가 2 로 푸른 실선의 원이 묘화 됩니다.   마지막으로 방금전 보존해 둔 원의 펜 핸들에 되돌리기 위해 재차SelectPen()를 호출해 원의 펜 핸들을 선택합니다.   　그런데 WTL는 스톡 펜도 서포트하고 있습니다. 이하에 나타내는 것은 스톡 펜에 의해 검은 실선의 타원을 묘화 하는 예입니다.   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 스톡 펜을 선택  HPEN hOldPen = dc.SelectStockPen(BLACK\_PEN);  // 클라이언트 구형을 취득해 , 엔을 묘화  CRect rect;  GetClientRect(rect);  dc.Ellipse(rect);  // 원의 펜을 선택  dc.SelectPen(hOldPen);  } |   그리고 ,atlmisc.h헤더에 정의되고 있다AtlGetStockPen()라고 하는 글로벌 함수를 사용하면 스톡 펜의 핸들을 취득할 수가 있습니다.   |  | | --- | | // 스톡 펜의 핸들을 취득  HPEN hStockPen = AtlGetStockPen(BLACK\_PEN); | |

**▣ GDI > 폰트**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WTL은 GDI오브젝트의 브러쉬를CBrushT이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는bool값으로 true의 경우는 소멸자로DeleteObject()를 호출해 false의 경우는 호출하지 않습니다. atlgdi.h헤더에서는typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlgdi.h내  typedef CBrushT<false> CBrushHandle;  typedef CBrushT<true> CBrush; |   　WTL의CBrush클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CBrush클래스를 사용해 클라이언트 구형에 접하는 푸른 격자 해치의 타원을 묘화 하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl04-03/01.png   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 푸른 격자 해치의 브러쉬를 작성해 선택  CBrush brush;  brush.CreateHatchBrush(HS\_CROSS, RGB(0, 0, 255));  HBRUSH hOldBrush = dc.SelectBrush(brush);  // 클라이언트 구형을 취득해 타원을 묘화  CRect rect;  GetClientRect(rect);  dc.Ellipse(rect);  // 원의 브러쉬를 선택  dc.SelectBrush(hOldBrush);  } |   　이 예에서는 우선 브러쉬 오브젝트를 작성해 그것을SelectBrush()로 현재의 브러쉬로서 선택하고 있습니다. SelectBrush()는 인수에HBRUSH형태의 변수를 받습니다만 , CBrushT클래스는 HBRUSH 변환 연산자를 가지고 있으므로 그대로 건네줄 수가 있습니다. SelectBrush()은 이전의 브러쉬 핸들을 돌려주므로 이것을hOldBrush 이라고 하는 변수에 보존해 둡니다.   다음에 타원을 묘화 합니다. 푸른 격자 해치의 타원이 묘화 됩니다.   마지막으로 방금전 보존해 둔 원의 브러쉬 핸들에 되돌리기 때문에 재차SelectBrush()를 호출해 원의 브러쉬 핸들을 선택합니다.   　그런데 WTL는 스톡 브러쉬도 서포트하고 있습니다. 이하에 나타내는 것은 스톡 브러쉬에 의해 검게 전부 칠해진 타원을 묘화 하는 예입니다.   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 스톡 브러쉬를 선택  HBRUSH hOldBrush = dc.SelectStockBrush(BLACK\_BRUSH);  // 클라이언트 구형을 취득해 , 엔을 묘화  CRect rect;  GetClientRect(rect);  dc.Ellipse(rect);  // 원의 브러쉬를 선택  dc.SelectBrush(hOldBrush);  } |   그리고 atlmisc.h헤더에 정의되고 있다 AtlGetStockBrush()라고 하는 글로벌 함수를 사용하면 스톡 브러쉬의 핸들을 취득할 수가 있습니다.   |  | | --- | | // 스톡 브러쉬의 핸들을 취득  HBRUSH hStockBrush = AtlGetStockBrush(BLACK\_BRUSH); | |

**▣ GDI > 파렛트**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| WTL은 GDI오브젝트의 폰트를 CFontT이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는 bool값으로 true의 경우는 소멸자로DeleteObject()를 호출해 false의 경우는 호출하지 않습니다. atlgdi.h헤더에서는typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlgdi.h내  typedef CFontT<false> CFontHandle;  typedef CFontT<true> CFont; |   　WTL의 CFont클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 아래에 나타내는 것은 CFont클래스를 사용해 클라이언트 영역의 중앙에 "Arial Black"폰트의 캐릭터 라인을 묘화 하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl04-04/01.png   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 사이즈20포인트의"Arial Black"폰트를 작성해 , 선택  CFont font;  font.CreatePointFont(200, \_T("Arial Black"));  HFONT hOldFont = dc.SelectFont(font);  // 텍스트색을 빨강으로 설정  dc.SetTextColor(RGB(255, 0, 0));  // 클라이언트 구형을 취득해 , 캐릭터 라인을 묘화  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  } |   　이 예에서는 우선 폰트 오브젝트를 작성해 그것을SelectFont()로 현재의 폰트로서 선택하고 있습니다. 덧붙여CreatePointFont()의 제1인수는1/10포인트 단위의 font size를 받으므로 묘화 하고 싶은 포인트 사이즈를10배가 되어 건네줍니다. SelectFont()(은)는 인수에HFONT형태의 변수를 받습니다만 CFontT클래스는 HFONT변환 연산자를 가지고 있으므로 그대로 건네줄 수가 있습니다. SelectFont()는 이전의 폰트 핸들을 돌려주므로 이것을hOldFont이라고 하는 변수에 보존해 둡니다.   다음에 캐릭터 라인을 묘화 합니다. 폰트명이 Arial Black 로 font size가20포인트의 붉은 캐릭터 라인이 묘화 됩니다.   마지막으로 , 방금전 보존해 둔 원의 폰트 핸들에 되돌리기 위해 재차SelectFont()를 호출해 원의 폰트 핸들을 선택합니다.   　그런데 WTL는 스톡 폰트도 서포트하고 있습니다. 이하에 나타내는 것은 스톡 폰트에 의해 시스템 폰트로 캐릭터 라인을 묘화 하는 예입니다.   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 스톡 폰트를 선택  HFONT hOldFont = dc.SelectStockFont(SYSTEM\_FONT);  // 텍스트색을 빨강으로 설정  dc.SetTextColor(RGB(255, 0, 0));  // 클라이언트 구형을 취득해 , 캐릭터 라인을 묘화  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  } |   그리고 atlmisc.h 헤더에 정의되고 있다 AtlGetStockFont()라고 하는 글로벌 함수를 사용하면 스톡 폰트의 핸들을 취득할 수가 있습니다.   |  | | --- | | // 스톡 폰트의 핸들을 취득  HFONT hStockFont = AtlGetStockFont(SYSTEM\_FONT); |   게다가 atlapp.h헤더에 정의되고 있다. AtlGetDefaultGuiFont()라고 하는 글로벌 함수를 사용하면 , 스톡 폰트 중에서도 디폴트 GUI폰트의 핸들을 취득할 수가 있습니다.   |  | | --- | | // 디폴트GUI폰트의 핸들을 취득  HFONT hGUIFont = AtlGetDefaultGuiFont();  // AtlGetStockFont()을 사용해 위와 같은 폰트의 핸들을 취득  HFONT hStockFont = AtlGetStockFont(DEFAULT\_GUI\_FONT); |   디폴트 GUI폰트란 메뉴나 다이알로그 박스에 사용되는 디폴트의 폰트입니다. |

▣ GDI > 비트맵

|  |  |  |
| --- | --- | --- |
| WTL은 GDI오브젝트의 비트맵을 CBitmapT이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는bool값으로 true의 경우는 소멸자로 DeleteObject()를 호출해 false의 경우는 호출하지 않습니다. atlgdi.h헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlgdi.h내  typedef CBitmapT<false> CBitmapHandle;  typedef CBitmapT<true> CBitmap; |   　WTL의CBitmap클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있는 위 DIB도 서포트하고 있습니다. 아래에 나타내는 것은 CBitmap클래스를 사용해 파일 정보를 바탕으로 팔레트를 최적화해 클라이언트 영역의 중앙에 비트 맵 화상을 표시하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl04-06/01.png   |  | | --- | | CBitmap m\_bitmap; // CBitmap형태의 멤버 변수  CPalette m\_palette; // CPalette형의 멤버 변수  // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 팔레트를 선택  HPALETTE hOldPalette;  if(!m\_palette.IsNull()){  hOldPalette = dc.SelectPalette(m\_palette, FALSE);  dc.RealizePalette();  }  if(!m\_bitmap.IsNull()){  // 메모리데바이스콘테키스트를 작성해 , 비트 맵을 선택  CDC dcMem;  dcMem.CreateCompatibleDC(dc);  HBITMAP hOldBitmap = dcMem.SelectBitmap(m\_bitmap);  // 클라이언트 구형과 화상 구형을 취득해 , 중앙에 화상을 표시  CRect rect;  GetClientRect(rect);  DIBSECTION ds;  ::GetObject(m\_bitmap, sizeof(DIBSECTION), &ds);  int nX = rect.Width() / 2 - ds.dsBm.bmWidth / 2;  int nY = rect.Height() / 2 - ds.dsBm.bmHeight / 2;  dc.BitBlt(nX, nY, ds.dsBm.bmWidth, ds.dsBm.bmHeight, dcMem, 0, 0, SRCCOPY);  // 원의 비트 맵을 선택  dcMem.SelectBitmap(hOldBitmap);  }  // 원의 팔레트를 선택  if(!m\_palette.IsNull())  dc.SelectPalette(hOldPalette, FALSE);  }  // WM\_CREATE메세지 핸들러  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 비트 맵 파일 읽기  m\_bitmap = AtlLoadBitmapImage(\_T("D:\\sample.bmp"),  LR\_LOADFROMFILE | LR\_CREATEDIBSECTION);  CClientDC dc(m\_hWnd);  if(dc.GetDeviceCaps(RASTERCAPS) & RC\_PALETTE){  // 화상 파일 정보를 취득  DIBSECTION ds;  ::GetObject(m\_bitmap, sizeof(DIBSECTION), &ds);  // 화상의 색 가지수를 취득  int nColors;  if(ds.dsBmih.biClrUsed != 0)  nColors = ds.dsBmih.biClrUsed;  else  nColors = 1 << ds.dsBmih.biBitCount;  // 화상에 적절한 팔레트를 작성  if(nColors > 256){  // 색 가지수가256색보다 많은 경우는 하프톤 팔레트를 작성  m\_palette.CreateHalftonePalette(dc);  }else{  // 색 가지수가256색 이하의 경우는 커스텀 팔레트를 작성  RGBQUAD\* pRGB = new RGBQUAD[nColors];  CDC dcMem;  dcMem.CreateCompatibleDC(dc);  HBITMAP hOldBitmap = dcMem.SelectBitmap(m\_bitmap);  dcMem.GetDIBColorTable(0, nColors, pRGB);  dcMem.SelectBitmap(hOldBitmap);  UINT nSize = sizeof(LOGPALETTE) + (sizeof(PALETTEENTRY) \* (nColors - 1));  LOGPALETTE\* pLP = (LOGPALETTE\*) new BYTE[nSize];  pLP->palVersion = 0x300;  pLP->palNumEntries = nColors;  for(int i=0; i<nColors; i++){  pLP->palPalEntry[i].peRed = pRGB[i].rgbRed;  pLP->palPalEntry[i].peGreen = pRGB[i].rgbGreen;  pLP->palPalEntry[i].peBlue = pRGB[i].rgbBlue;  pLP->palPalEntry[i].peFlags = 0;  }  m\_palette.CreatePalette(pLP);  delete[] pLP;  delete[] pRGB;  }  }  return 0;  } |   　우선, 선두에서 CBitmap클래스의 인스턴스인m\_bitmap과 CPalette클래스의 인스턴스인m\_palette을 멤버 변수로서 선언합니다. 이러한 변수는 각각CBitmap,CPalette클래스의 constructor으로 NULL에 초기화되고 있습니다.   　다음에 WM\_CREATE메세지 핸들러를 봅니다. WM\_CREATE메세지 핸들러에서는 초에 AtlLoadBitmapImage()에 의해 비트 맵 파일을 읽어들입니다. 다음에 팔레트가 필요한지 어떤지를 체크해 필요한 경우는 화상의 색 가지수를 취득해 적절한 팔레트를 작성합니다.   　WM\_PAINT메세지 핸들러에서는 우선 필요한 경우는 팔레트 오브젝트를 선택합니다. 다음에 메모리 디바이스 컨텍스트를 작성해SelectBitmap()로 비트맵 오브젝트를 선택합니다. SelectBitmap()은 인수에 HBITMAP형태의 변수를 받습니다만 CBitmapT클래스는HBITMAP 변환 연산자를 가지고 있으므로 그대로 건네줄 수가 있습니다. SelectBitmap()은 이전의 비트 맵 핸들을 돌려주므로 이것을hOldBitmap이라고 하는 변수에 보존해 둡니다.   다음에 클라이언트 영역의 중앙에 비트 맵을 묘화 합니다. 묘화 하는 좌표는 GetClientRect()이라고GetObject()에 의해 각각 클라이언트 영역과 화상의 크기를 취득해 계산합니다.   마지막으로 방금전 보존해 둔 원의 비트 맵 핸들에 되돌리기 위해 재차SelectBitmap()를 호출해 원의 비트 맵 핸들을 선택합니다. 팔레트를 사용했을 경우는 팔레트도 바탕으로 되돌립니다. |
|  |

▣ GDI > 리젼

WTL은 GDI오브젝트의 리젼을 CRgnT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는bool값으로 true의 경우는 소멸자로 DeleteObject()를 호출해 false의 경우는 호출하지 않습니다. atlgdi.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlgdi.h내  typedef CRgnT<false> CRgnHandle;  typedef CRgnT<true> CRgn; |

　WTL의 CRgn클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 아래에 나타내는 것은 CRgn클래스를 사용해 클라이언트 영역의 중앙에 "Arial Black"폰트의 캐릭터 라인을 묘화 하는 예입니다.



|  |
| --- |
| // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  // 사이즈 20 포인트의 "Arial Black" 폰트를 작성해 선택  CFont font;  font.CreatePointFont(200, \_T("Arial Black"));  HFONT hOldFont = dc.SelectFont(font);  // 패스를 작성  dc.BeginPath();  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  dc.EndPath();  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  CRgn rgn1;  rgn1.CreateFromPath(dc);  CRgn rgn2;  rgn2.CreateRectRgnIndirect(rect);  // 리젼을 합성  rgn1.CombineRgn(rgn1, rgn2, RGN\_XOR);  // 리젼을 클리핑 영역으로서 선택  dc.SelectClipRgn(rgn1);  // 굵기1로 초록의 실선의 펜을 작성해 , 선택  CPen pen;  pen.CreatePen(PS\_SOLID, 1, RGB(0, 128, 0));  HPEN hOldPen = dc.SelectPen(pen);  // 섬 모양을 묘화  for(int i=0; i<rect.bottom; i+=2){  dc.MoveTo(0, i);  dc.LineTo(rect.right, i);  }  // 원의 펜을 선택  dc.SelectPen(hOldPen);  } |

　이 예에서는 우선 폰트를 작성해 캐릭터 라인을 묘화 해 그것을 바탕으로 패스를 작성합니다. 그 패스로부터 리젼을 작성해 한층 더 클라이언트 구형으로부터 리젼을 작성해 이 두 개의 리젼을 CombineRgn()에 의해 합성합니다. CombineRgn()의 제3인수에는RGN\_XOR을 지정해 있으므로 rgn1(캐릭터 라인 리젼)rgn2(클라이언트 구형 리젼) 를 합한 것으로부터 서로 겹친 부분을 없앤 리젼이 작성됩니다.   
  
다음에 그 합성한 리젼을 클리핑 영역으로서 선택해 클라이언트 영역에 초록의 섬 모양을 묘화 합니다. 이 때, 클리핑 영역으로서 선택된 리젼 부분에는 모양이 묘화 되지 않기 때문에 흰 배경이 그대로 보입니다.

▣ GDI > 확장 메타 파일

|  |  |  |
| --- | --- | --- |
| WTL은 확장 메타파일의 기능을 CEnhMetaFileT이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는bool값으로 true의 경우는 소멸자로 DeleteEnhMetaFile()를 호출해 false의 경우는 호출하지 않습니다. atlgdi.h헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlgdi.h안  typedef CEnhMetaFileT<false> CEnhMetaFileHandle;  typedef CEnhMetaFileT<true> CEnhMetaFile; |   그리고 WTL는 확장 메타파일용 디바이스 콘텍스트를CEnhMetaFileDC클래스, 확장 메타파일 정보를 CEnhMetaFileInfo 클래스에서 캡슐화하고 있습니다.   　이하에 나타내는 것은 WTL의 확장 메타파일 서포트 클래스를 사용해, 확장 메타파일을 작성 묘화 하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl04-08/01.png   |  | | --- | | // WM\_PAINT메세지 핸들러  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CEnhMetaFile emf(::GetEnhMetaFile(\_T("test.emf")));  CRect rect;  GetClientRect(rect);  // 확장 메타파일을 묘화  dc.PlayMetaFile(emf, rect);  }  // WM\_CREATE메세지 핸들러  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 확장 메타파일을 작성  CEnhMetaFileDC dc(NULL, \_T("test.emf"), NULL, NULL);  // 푸른 격자 해치의 브러쉬를 작성해 선택  CBrush brush;  brush.CreateHatchBrush(HS\_CROSS, RGB(0, 0, 255));  HBRUSH hOldBrush = dc.SelectBrush(brush);  // 클라이언트 구형을 취득해 타원을 묘화  CRect rect;  GetClientRect(rect);  dc.Ellipse(rect);  // 원의 브러쉬를 선택  dc.SelectBrush(hOldBrush);  return 0;  } |   　이 예에서는 우선WM\_CREATE메세지 핸들러로 경향 폴더에 확장 메타파일을 작성합니다. 이번 사용한CEnhMetaFileDC클래스의 인수 있어 constructor은 내부에서Win32API의CreateEnhMetaFile()를 호출합니다만 디폴트 constructor 을 사용했을 경우는 굳이 호출하지 않습니다. 그 경우는 CEnhMetaFileDC클래스의 멤버 함수인 Create()을 호출하는 것으로 확장 메타파일을 작성할 수 있습니다. 확장 메타파일을 작성한 뒤는 동시에 얻을 수 있는 확장 메타파일용 디바이스 컨텍스트를 사용해 도형을 묘화 합니다.   덧붙여 CEnhMetaFileDC클래스의 소멸자에서는CloseEnhMetaFile()이 불려 가기 때문에, 확장 메타파일용 디바이스 콘텍스트를 명시적으로 닫을 필요는 없습니다. Close()를 호출하면 명시적으로 닫을 수가 있어 반환값으로서 확장 메타파일의 핸들을 취득할 수 있습니다.   　WM\_PAINT메세지 핸들러에서는 Win32API의 GetEnhMetaFile()를 호출해 경향 폴더로부터 방금 전 작성한 확장 메타파일을 읽어들여 PlayMetaFile() 그리고 확장 메타파일의 내용을 묘화 합니다. |

▣ 메뉴 > 메뉴 바

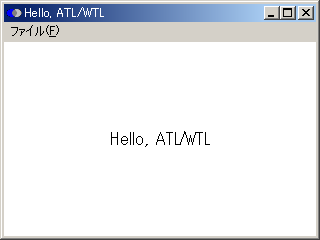
|  |  |  |  |
| --- | --- | --- | --- |
| 지금까지의 프로그램에는 도구모음이 없었습니다. 여기에서는 CWndClassInfo구조체를 사용해 「Hello, ATL/WTL」프로그램에 도구모음을 추가합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl05-01/01.png   |  | | --- | | // stdafx.h내  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h> |  |  | | --- | | // MainWindow.h내  class CMyWindow : public CWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler  {  public:  // 윈도우 클래스명 , 도구모음을 등록  static CWndClassInfo& GetWndClassInfo()  {  static CWndClassInfo wc =  {  {sizeof(WNDCLASSEX), CS\_HREDRAW | CS\_VREDRAW, StartWindowProc,  0, 0, NULL, NULL, NULL, (HBRUSH)(COLOR\_WINDOW + 1),  MAKEINTRESOURCE(IDR\_MENU\_MAIN), // 도구모음  "Hello", NULL},  NULL, NULL, IDC\_ARROW, TRUE, 0, \_T("")  };  return wc;  }  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 아이돌 처리  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_DESTROY(OnDestroy)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnDestroy(){  PostQuitMessage(0);  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // hello.cpp내  #include "stdafx.h"  #include "resource.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.Create(NULL, CWindow::rcDefault,  \_T("Hello, ATL/WTL"), WS\_OVERLAPPEDWINDOW | WS\_VISIBLE);  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　우선 도구모음을 위한 메뉴 자원을 프로젝트에 추가합니다. 메뉴 자원의 것ID은 IDR\_MENU\_MAIN으로 해 톱 레벨에[파일]을 그 아래에[종료]라고 하는 메뉴 아이템을 추가합니다. [종료]메뉴 아이템에는ID\_MENUITEM\_EXIT이라고 하는ID을 설정합니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl05-01/02.png  다음에 프로젝트내의 resource file(확장자(extension).rc) 를 메모장등의 텍스트 문자 편집기로 열어 「afxres.h」이라고 하는 문자를 모두 「atlres.h」에 옮겨놓아 덧쓰기 보존합니다. 그러면 ,Visual C++ 에서 다음과 같은 다이얼로그가 표시되므로 [네]를 선택합니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl05-01/03.png  덧붙여ATL/WTL AppWizard 로 작성했을 경우는, 이와 같이 resource file를 직접 텍스트 문자 편집기로 편집할 필요는 없습니다.   　CMyWindow클래스에서는 ,CWndClassInfo구조체를 커스터마이즈 하는 것에 의해 "Hello"라고 하는 윈도우 클래스명과 자원ID이IDR\_MENU\_MAIN의 메뉴 자원을 윈도우에 등록하고 있습니다.   메세지 맵에는 ID이ID\_MENUITEM\_EXIT의 커멘드 메세지용의 엔트리를 추가해 OnMenuExit()라고 하는 메세지 핸들러를 호출하도록 하고 있습니다.   　hello.cpp안에서는 ,resource.h헤더를 인클루드 합니다. |

▣ 메뉴 > 팝업 메뉴

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| WTL에서 메뉴를CMenuT이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는bool값으로 true의 경우는 소멸자로DestroyMenu()를 호출해 false의 경우는 호출하지 않습니다. atluser.h헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atluser.h내  typedef CMenuT<false> CMenuHandle;  typedef CMenuT<true> CMenu; |   　WTL의CMenu클래스는MFC의 동명의 클래스와 동등의 기능을 갖추고 있습니다. 여기에서는 CMenu클래스를 사용해 전회의 도구모음을 추가한 「Hello, ATL/WTL」프로그램에 pop-up menu를 추가합니다. 이 예에서는 , 클라이언트 영역에서 오른쪽 클릭했을 때에 아래의 그림과 같은 pop-up menu가 표시되도록 합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl05-02/01.png   |  | | --- | | // stdafx.h내  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h> |  |  | | --- | | // MainWindow.h내  class CMyWindow : public CWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler  {  public:  // 윈도우 클래스명 도구모음을 등록  static CWndClassInfo& GetWndClassInfo()  {  static CWndClassInfo wc =  {  {sizeof(WNDCLASSEX), CS\_HREDRAW | CS\_VREDRAW, StartWindowProc,  0, 0, NULL, NULL, NULL, (HBRUSH)(COLOR\_WINDOW + 1),  MAKEINTRESOURCE(IDR\_MENU\_MAIN), // 도구모음  "Hello", NULL},  NULL, NULL, IDC\_ARROW, TRUE, 0, \_T("")  };  return wc;  }  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 아이들 처리  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_DESTROY(OnDestroy)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLO, OnMenuHello)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnDestroy(){  PostQuitMessage(0);  }  void OnMenuHello(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello, ATL/WTL"));  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // hello.cpp내  #include "stdafx.h"  #include "resource.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.Create(NULL, CWindow::rcDefault,  \_T("Hello, ATL/WTL"), WS\_OVERLAPPEDWINDOW | WS\_VISIBLE);  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　우선 pop-up menu를 위한 메뉴 자원을 프로젝트에 추가합니다. 메뉴 자원의 것ID은 IDR\_MENU\_POPUP으로 해 톱 레벨에[index0]를 그 아래에[Hello]메뉴 아이템과 [종료]메뉴 아이템을 추가합니다. [Hello]메뉴 아이템에는ID\_MENUITEM\_HELLO, [종료]메뉴 아이템에는ID\_MENUITEM\_EXIT이라고 하는ID을 설정합니다. 덧붙여ID\_MENUITEM\_EXIT라고 하는ID은 벌써 전회 작성한 것입니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl05-02/02.png  　CMenu클래스는atluser.h헤더에 정의되고 있습니다만 이 헤더는atlapp.h헤더를 인클루드 한 시점에서 자동적으로 인클루드 되고 있습니다.   　CMyWindow클래스에는 우선WM\_CONTEXTMENU메세지 핸들러를 추가해 그런데 pop-up menu 자원을 로드해TrackPopupMenu()에 의해 pop-up menu를 표시시키고 있습니다.   　다음에 pop-up menu 아이템이 선택되었을 때의 커멘드 메세지 핸들러를 추가합니다. 메세지 맵에ID가ID\_MENUITEM\_HELLO의 커멘드 메세지용의 엔트리를 추가해 OnMenuHello()라고 하는 메세지 핸들러를 호출하도록 합니다. 덧붙여 자원ID이ID\_MENUITEM\_EXIT의 커멘드 메세지 핸들러는 벌써 전회 작성하였습니다.  　그런데 위의 예에서는 메뉴 자원을 로드해 메뉴를 작성했습니다만 메뉴는 동적으로 작성할 수도 있습니다. 아래에 나타내는 것은 위의 예와 같은 pop-up menu를 동적으로 작성하는 예입니다.   |  | | --- | | void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.CreatePopupMenu();  menuPopup.AppendMenu(MF\_STRING, ID\_MENUITEM\_HELLO, \_T("Hello(&H)"));  menuPopup.AppendMenu(MF\_SEPARATOR);  menuPopup.AppendMenu(MF\_STRING, ID\_MENUITEM\_EXIT, \_T("종료(&X)"));  menuPopup.TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  } | |

▣ 프레임 윈도우 > 프레임 윈도우

대부분의 메인 윈도우에는 도구모음이 있어 WM\_DESTROY메세지 핸들러에서는 단지PostQuitMessage()를 호출하고 있습니다. 그런데 지금까지 처럼 CWindowImpl클래스를 메인 윈도우의 base class로서 사용하면, 매회 스스로CWndClassInfo구조체를 커스터마이즈 해 메뉴 자원을 설정하거나 메세지 맵에WM\_DESTROY메세지용의 엔트리를 추가해 PostQuitMessage()를 호출하기 위해 WM\_DESTROY메세지 핸들러를 쓰지 않으면 안됩니다.   
  
　WTL은 이러한 기본적인 기능을 가지는 윈도우를 간단하게 만들기 위해서 CFrameWindowImpl라고 하는 클래스를 준비해 있습니다. CFrameWindowImpl(또는 그 base class이다 FrameWindowImplBase) 클래스에는 WM\_DESTROY메세지용의 디폴트의 핸들러가 준비되어 있는 것 외 윈도우 사이즈가 변경되었을 때에 자동적으로 레이아웃을 조정하는WM\_SIZE메세지 핸들러 툴바나 상태 바를 작성하는 함수 키보드 가속기용의 코드나 툴 팁용의 메세지 핸들러등이 갖춰져 있습니다. 그리고 도구모음등의 자원을 간단하게 설정하는 매크로를 서포트하고 있습니다.   
  
　여기에서는 전회의 pop-up menu를 추가한 「Hello, ATL/WTL」프로그램의 base class를 CWindowImpl로부터CFrameWindowImpl에 변경해 고쳐 씁니다.



|  |
| --- |
| // stdafx.h안  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // CFrameWindowImpl클래스를 사용하기 위해(때문에) |

|  |
| --- |
| // MainWindow.h안  // 메인 윈도우의 base class를CFrameWindowImpl로 변경  class CMyWindow : public CFrameWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler  {  public:  // 윈도우 클래스명 , 공통 자원ID, 스타일 , 배경색을 등록  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("Hello"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  // base class의 것PreTranslateMessage을 호출하는  return CFrameWindowImpl<CMyWindow>::PreTranslateMessage(pMsg);  }  // 아이돌 처리  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLO, OnMenuHello)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuHello(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello, ATL/WTL"));  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

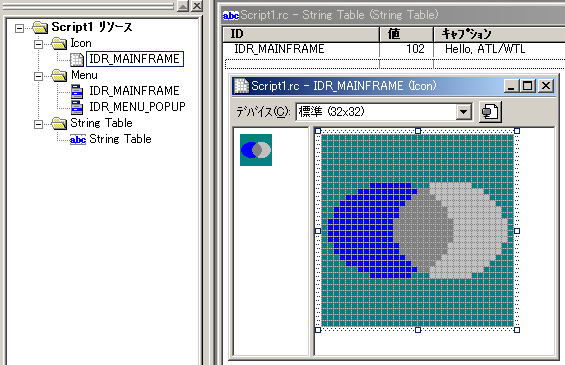
|  |
| --- |
| // hello.cpp내  #include "stdafx.h"  #include "resource.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |

　우선 stdafx.h안에서는CFrameWindowImpl클래스를 사용하기 위해서atlframe.h헤더를 인클루드 합니다.   
  
　다음에 , 메인 윈도우인CMyWindow클래스의 base class를 CWindowImpl클래스에서CFrameWindowImpl클래스에 변경합니다. CFrameWindowImpl클래스도CWindowImpl클래스때와 같게 , 3개의 템플릿 인수를 가집니다. 여기에서는 생략 가능한 제2, 3템플릿 인수를 생략 해 제1템플릿 인수에 상속 클래스의 이름을 건네줍니다. 덧붙여 디폴트로 제2템플릿 인수에는CWindow, 제3템플릿 인수에는 윈도우 특성으로서CFrameWinTraits가 건네받습니다. CFrameWinTraits(은)는ATL의atlwin.h헤더로 다음과 같이 정의되고 있습니다.

|  |
| --- |
| // atlwin.h내  typedef CWinTraits<WS\_OVERLAPPEDWINDOW | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS,  WS\_EX\_APPWINDOW | WS\_EX\_WINDOWEDGE> CFrameWinTraits; |

　CMyWindow클래스에서는 우선 프레임 윈도우용의 윈도우 클래스명 등록 매크로를 사용해 윈도우 클래스명 , 자원ID, 스타일 , 배경색을 등록하고 있습니다. 이 때 자원ID은 이하의 자원으로 공통입니다.

* 윈도우 아이콘
* 도구모음
* 윈도우 타이틀 캐릭터 라인
* 키보드 가속기
* 툴바

즉 , 이러한 자원에 공통의 자원ID을 붙여 그ID를DECLARE\_FRAME\_WND\_CLASS\_EX매크로에 의해 등록하면 CFrameWindowImpl클래스는 그러한 자원을 자동적으로 윈도우로 설정합니다. 이번 예에서는 공통 자원ID으로서IDR\_MAINFRAME를 사용하기로 했습니다. 그런데 아이콘 자원이나 윈도우 타이틀을 위한 캐릭터 라인 자원을 추가해 각각IDR\_MAINFRAME이라고 하는ID을 붙입니다. 그리고 도구모음을 위한 메뉴 자원ID도IDR\_MAINFRAME로 변경합니다.   
  
  
  
덧붙여 이번 예에서는 툴바나 키보드 가속기는 사용하지 않기 때문에 그것을 위한 자원은 추가하고 있지 않습니다.   
  
　다음에 메세지 필터에서는 base class인CFrameWindowImpl클래스의 것 PreTranslateMessage()을 호출합니다. CFrameWindowImpl::PreTranslateMessage()는 키보드 가속기용으로 ::TranslateAccelerator()를 호출하고 있습니다. 이하는 그 소스입니다.

|  |
| --- |
| // atlframe.h안  BOOL PreTranslateMessage(MSG\* pMsg)  {  if(m\_hAccel != NULL && ::TranslateAccelerator(m\_hWnd, m\_hAccel, pMsg))  return TRUE;  return FALSE;  } |

다만 이번은 키보드 가속기를 사용하지 않기 때문에 너무 관계 없습니다.   
  
　다음에 메세지 맵에 base class인CFrameWindowImpl클래스에 체인을 추가합니다. 이것에 의해 WM\_DESTROY메세지에 대한 처리는 CMyWindow클래스가 아니고 CFrameWindowImpl클래스에 맡길 수가 있으므로 상기의 원시 코드로부터는 WM\_DESTROY메세지 핸들러용의 코드를 삭제하고 있습니다.   
  
　물론 스스로WM\_DESTROY메세지 핸들러를 준비해 거기에 자신의 어플리케이션 특유의 처리를 더할 수가 있습니다. 그 경우 base class인 CFrameWindowImpl클래스에 메세지를 보내는 것을 잊어서는 안됩니다. 그렇게 하지 않으면 최종적으로 ::PostQuitMessage()가 불려 가지 않고 프로그램이 종료하지 않기 때문입니다. 이하는 스스로WM\_DESTROY메세지 핸들러를 준비했을 경우의 코드입니다.

|  |
| --- |
| // MainWindow.h안  class CMyWindow : public CFrameWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler, public CUpdateUI<CMyWindow>  {  ...    // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_DESTROY(OnDestroy)  ...  ...  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  END\_MSG\_MAP()  ...  ...    void OnDestroy(){  // 어플리케이션 특유의 처리  ...    // 이 함수 호출에 의해 메세지는 체인처의 것CFrameWindowImpl에 보내진다  SetMsgHandled(false);  }  } |

　이 예에서는 우선 WM\_DESTROY메세지가 보내져 오면 CMyWindow클래스의 메세지 맵의MSG\_WM\_DESTROY()엔트리에 의해 CMyWindow::OnDestroy()가 실행됩니다. CMyWindow::OnDestroy()그럼 어플리케이션 특유의 처리를 하고 나서 SetMsgHandled(false)를 호출합니다. 이 호출에 의해 한층 더 메세지 맵내가 검색되어 WM\_DESTROY메세지는 체인에 의해 base class인CFrameWindowImpl클래스에 보내집니다. CFrameWindowImpl클래스내에서는 한층 더 체인에 의해 base class인CFrameWindowImplBase클래스에 보내져 최종적으로CFrameWindowImplBase클래스의WM\_DESTROY메세지 핸들러로 ::PostQuitMessage()가 불려 가 어플리케이션은 종료합니다.   
  
　\_tWinMain()는 윈도우를 작성하기 위해서CreateEx()를 사용하고 있습니다. CFrameWindowImpl클래스는Create()도 준비해 있습니다만 공통 자원ID에 의해 자동적으로 자원을 설정하려면 CreateEx()를 사용할 필요가 있습니다.

▣ 프레임 윈도우 > 프레임 윈도우 클래스 정보

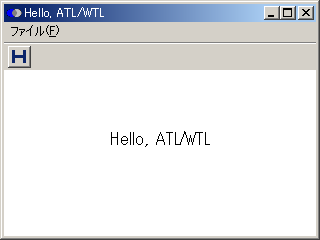
|  |
| --- |
| 메인 윈도우의 base class에CWindowImpl클래스를 사용하고 있었을 경우는 윈도우 클래스명이나 배경색, 커서 등 속성을 지정하기 위해서DECLARE\_WND\_CLASS등의 매크로나 CWndClassInfo구조체를 사용했습니다. 메인 윈도우의 base class에 CFrameWindowImpl클래스를 사용했을 경우에도 같은 매크로나 클래스가 준비되어 있습니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 프레임 윈도우 클래스명 매크로 |
| 프레임 윈도우 클래스명을 지정하기 위한 매크로를 이하에 나타냅니다.   * DECLARE\_FRAME\_WND\_CLASS(윈도우 클래스명, 공통 자원ID)  윈도우 클래스명과 공통 자원ID을 정의합니다. 덧붙여ATL의DECLARE\_WND\_CLASS매크로에서는 디폴트의 스타일로서 CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS 를 설정해 줍니다만 , DECLARE\_FRAME\_WND\_CLASS그럼 0 을 설정합니다. * DECLARE\_FRAME\_WND\_CLASS\_EX(윈도우 클래스명, 공통 자원ID, 스타일, 배경색)  윈도우 클래스명 , 공통 자원ID, 스타일 , 배경색을 정의합니다.   이러한 매크로는public선언으로 사용하지 않으면 안됩니다. 이러한 매크로를 사용해 윈도우 클래스명을 명시적으로 등록하지 않는 또는 윈도우 클래스명으로서NULL를 지정했을 경우 WTL는 "ATL:00406060" 과 같은 이름을 자동적으로 등록합니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | CFrameWndClassInfo클래스 |
| CFrameWndClassInfo클래스는 전술의 매크로보다 프레임 윈도우의 속성을 세세하게 지정할 수 있습니다. CFrameWndClassInfo클래스를 사용해 새로운 속성을 지정하기 위해서는 CFrameWindowImpl::GetWndClassInfo()을 오버라이드(override) 해 커스터마이즈 한 정적인CFrameWndClassInfo인스턴스의 참조를 돌려줍니다.   |  | | --- | | class CMyWindow: public CFrameWndClassInfo<CMyWindow>  {  public:  static WTL::CFrameWndClassInfo& GetWndClassInfo()  {  static WTL::CFrameWndClassInfo wc =  {  {sizeof(WNDCLASSEX), CS\_HREDRAW | CS\_VREDRAW, StartWindowProc,  0, 0, NULL, NULL, NULL, (HBRUSH)(COLOR\_WINDOW + 1), NULL,  "MyWindow", NULL}, // WNDCLASSEX구조체  NULL, // 기존의 윈도우 클래스명  NULL, // 기존의 윈도우 프로시저  MAKEINTRESOURCE(IDC\_CURSOR1), // 커서 자원명  FALSE, // 시스템 커서라면TRUE, 그 이외는FALSE  0, // 등록이 끝난 윈도우 클래스의 식별자  \_T("") // ATL가 자동 생성한 윈도우 클래스명  IDR\_MAINFRAME // 공통 자원ID  };  return wc;  }  ...  }; | | |

▣ 프레임 윈도우 > 심플 툴바

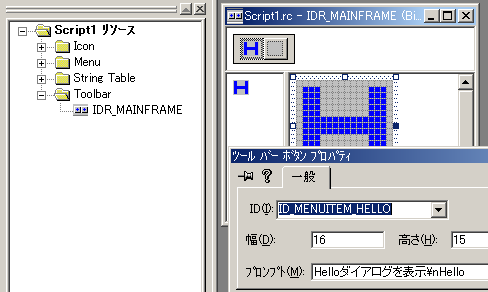
이번은 지금까지 작성한 「Hello, ATL/WTL」프로그램에 툴바를 추가합니다. CFrameWindowImpl클래스에는 간단한 툴바를 작성하기 위해서 CreateSimpleToolBar()라고 하는 멤버 함수가 준비되어 있습니다.



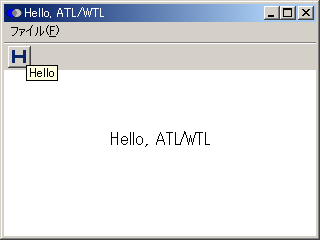
|  |
| --- |
| // stdafx.h안  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // CFrameWindowImpl클래스를 사용하기 위해 |

|  |
| --- |
| // MainWindow.h  // 메인 윈도우의 base class를CFrameWindowImpl로 변경  class CMyWindow : public CFrameWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler  {  public:  // 윈도우 클래스명 , 공통 자원ID, 스타일 , 배경색을 등록  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("Hello"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  // base class의 것PreTranslateMessage을 호출하는  return CFrameWindowImpl<CMyWindow>::PreTranslateMessage(pMsg);  }  // 아이돌 처리  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLO, OnMenuHello)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 툴바를 작성  CreateSimpleToolBar();  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuHello(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello, ATL/WTL"));  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // hello.cpp내  #include "stdafx.h"  #include "resource.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |

　우선은 프로젝트에 툴바 자원을 추가해 자원ID으로서IDR\_MAINFRAME를 설정합니다. 그리고 툴바에 버튼을 하나 추가해 그 버튼에ID\_MENUITEM\_HELLO라고 하는 자원ID을 설정합니다. 이것은 벌써 pop-up menu를 작성했을 때에 사용하고 있는ID입니다.   
  
  
  
　다음에 ,CMyWindow클래스OnCreate()중에서 CreateSimpleToolBar()를 호출합니다. 이만큼으로 메인 윈도우에 툴바가 추가되어 툴바 상의 버튼(ID：ID\_MENUITEM\_HELLO) 을 누르면(자) , OnMenuHello()가 불려 가게 됩니다.   
  
덧붙여 CreateSimpleToolBar()에는 3개의 인수를 건네줄 수가 있습니다. 제1인수에는 툴바 자원ID을 지정합니다. 디폴트 인수는 0 으로 이 경우는 공통 자원ID(이번 예에서는IDR\_MAINFRAME) 이 사용됩니다. 제2인수에는 툴바의 스타일을 지정합니다. 디폴트 인수는ATL\_SIMPLE\_TOOLBAR\_STYLE으로 이것은atlframe.h헤더로 다음과 같이 정의되고 있습니다.

|  |
| --- |
| // atlframe.h내  #define ATL\_SIMPLE\_TOOLBAR\_STYLE \  (WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS | TBSTYLE\_TOOLTIPS) |

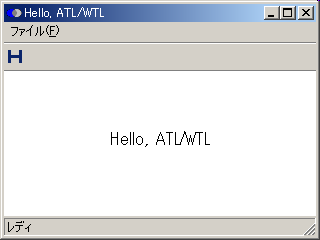
제3인수에는 식별자를 지정합니다. 디폴트 인수는ATL\_IDW\_TOOLBAR입니다.   
  
CreateSimpleToolBar()는 내부에서 CreateSimpleToolBarCtrl()라고 하는 멤버 함수를 호출하고 있어 그 호출에 의해 얻을 수 있던 툴바의 핸들을 CFrameWindowImpl 클래스의 base class인 CFrameWindowImplBase 클래스의 m\_hWndToolBar이라고 하는 HWND형태의 멤버 변수에 대입하고 있습니다.   
  
CFrameWindowImplBase클래스에는 TTN\_GETDISPINFOA,TTN\_GETDISPINFOW 메세지 핸들러가 준비되어 있습니다. 이 때문에 툴바의 버튼상에 커서를 실었을 때의TTN\_GETDISPINFOA(W)메세지가 체인처의 CFrameWindowImpl클래스에 보내지면 자동적으로 툴바 자원의[prompt]로 설정한 캐릭터 라인이 툴 팁으로 해서 표시됩니다.  
  


▣ 프레임 윈도우 > 심플 리바

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 여기에서는 전회의 툴바를 추가한 「Hello, ATL/WTL」프로그램에 리버를 추가해 그 리버의 밴드에 툴바를 격납합니다. CFrameWindowImpl클래스에는 간단한 리버를 작성하기 위한 멤버 함수가 준비되어 있습니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl06-04/01.png   |  | | --- | | // stdafx.h안  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // CFrameWindowImpl클래스를 사용하기 위해(때문에) |  |  | | --- | | // MainWindow.h  // 메인 윈도우의 base class를CFrameWindowImpl로 변경  class CMyWindow : public CFrameWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler  {  public:  // 윈도우 클래스명 , 공통 자원ID, 스타일 , 배경색을 등록  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("Hello"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  // base class의 것PreTranslateMessage을 호출하는  return CFrameWindowImpl<CMyWindow>::PreTranslateMessage(pMsg);  }  // 아이들 처리  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLO, OnMenuHello)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 리버를 작성  CreateSimpleReBar();  // 툴바를 작성해 밴드에 추가  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  AddSimpleReBarBand(hWndToolBar);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuHello(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello, ATL/WTL"));  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // hello.cpp내  #include "stdafx.h"  #include "resource.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  // 코먼 컨트롤 및 리버 초기화  AtlInitCommonControls(ICC\_WIN95\_CLASSES | ICC\_COOL\_CLASSES);  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　우선 CMyWindow클래스OnCreate()중에서 CreateSimpleReBar()를 호출해 리버를 작성합니다.   덧붙여 CreateSimpleReBar()에는 2개의 인수를 건네줄 수가 있습니다. 제1인수에는 리버의 스타일을 지정합니다. 디폴트 인수는ATL\_SIMPLE\_REBAR\_STYLE입니다. 리버의 스타일은 atlframe.h헤더로 다음과 같이 정의되고 있습니다. 덧붙여 표시상의 사정으로 개행 위치등을 변경하고 있습니다.   |  | | --- | | // atlframe.h내  // standard rebar styles  #if (\_WIN32\_IE >= 0x0400)  #define ATL\_SIMPLE\_REBAR\_STYLE \  (WS\_CHILD | WS\_VISIBLE | WS\_BORDER | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS | \  RBS\_VARHEIGHT | RBS\_BANDBORDERS | RBS\_AUTOSIZE)  #else  #define ATL\_SIMPLE\_REBAR\_STYLE \  (WS\_CHILD | WS\_VISIBLE | WS\_BORDER | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS | \  RBS\_VARHEIGHT | RBS\_BANDBORDERS)  #endif //!(\_WIN32\_IE >= 0x0400)  // rebar without borders  #if (\_WIN32\_IE >= 0x0400)  #define ATL\_SIMPLE\_REBAR\_NOBORDER\_STYLE \  (WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS | \  RBS\_VARHEIGHT | RBS\_BANDBORDERS | RBS\_AUTOSIZE | CCS\_NODIVIDER)  #else  #define ATL\_SIMPLE\_REBAR\_NOBORDER\_STYLE \  (WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS | \  RBS\_VARHEIGHT | RBS\_BANDBORDERS | CCS\_NODIVIDER)  #endif //!(\_WIN32\_IE >= 0x0400) |   제2인수에는 식별자를 지정합니다. 디폴트 인수는ATL\_IDW\_TOOLBAR입니다.   CreateSimpleReBar()은 내부에서CreateSimpleReBarCtrl()라고 하는 멤버 함수를 호출하고 있어 그 호출에 의해 얻을 수 있던 리버의 핸들을 CFrameWindowImpl클래스의 base class인 CFrameWindowImplBase 클래스의 m\_hWndToolBar이라고 하는 HWND 형태의 멤버 변수에 대입하고 있습니다.   　다음에 CreateSimpleToolBarCtrl()를 호출해 툴바를 작성합니다. CreateSimpleToolBarCtrl()의 제1인수는 메인 윈도우의 핸들 , 제2인수는 자원ID, 제3인수는 툴바의 선두에 separator를 설정하는지 어떤지의 플래그 , 제4인수에는 툴바의 스타일을 지정합니다. 제5인수에는 식별자를 지정할 수 있습니다만 이번은 생략 하고 있습니다. 리버에 밴드 하는 툴바의 경우는 스타일로서ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE 를 지정합니다. 이 스타일은atlframe.h헤더로 다음과 같이 정의되고 있습니다.   |  | | --- | | // atlframe.h내  // toolbar in a rebar pane  #define ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE \  (WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS | \  CCS\_NODIVIDER | CCS\_NORESIZE | CCS\_NOPARENTALIGN | TBSTYLE\_TOOLTIPS | TBSTYLE\_FLAT) |   　다음에 작성한 툴바를 AddSimpleReBarBand()에 의해 리버의 밴드에 추가합니다. 이번은 제1인수로서 툴바의 핸들 밖에 지정해 있었습니다만 , 제2인수에는 타이틀 , 제3인수에는 새로운 행에 두는지 어떤지의 플래그 , 제4인수에는 폭 , 제5인수에는 폭보다 작게 할 수 있도록 하는지 어떤지의 플래그를 설정할 수 있습니다.   　\_tWinMain()에서 리버를 초기화하기 위해서ICC\_COOL\_CLASSES플래그를 지정했던 AtlInitCommonControls()를 호출합니다. AtlInitCommonControls()은 Win32API의 InitCommonControlsEx()함수를 랩 합니다. 이번은 자주 사용된다고 생각되는ICC\_WIN95\_CLASSES플래그도 추가해 두었습니다. |

▣ 프레임 윈도우 > 심플 스테이터스 바

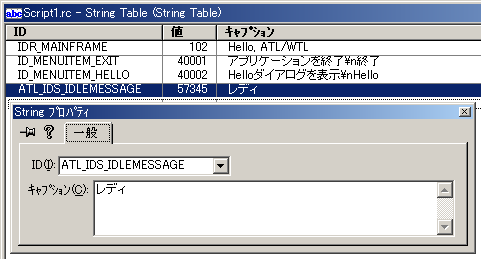
여기에서는 전회의 리버를 추가한 「Hello, ATL/WTL」프로그램에 상태 바를 추가합니다. CFrameWindowImpl클래스에는 간단한 상태 바를 작성하기 위해서 CreateSimpleStatusBar()라고 하는 멤버 함수가 준비되어 있습니다.



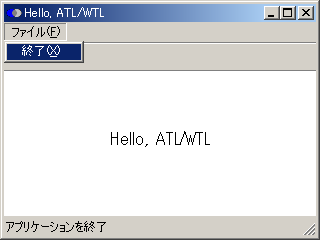
|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // CFrameWindowImpl클래스를 사용하기 위해(때문에) |

|  |
| --- |
| // MainWindow.h  // 메인 윈도우의 base class를CFrameWindowImpl로 변경  class CMyWindow : public CFrameWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler  {  public:  // 윈도우 클래스명 , 공통 자원ID, 스타일 , 배경색을 등록  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("Hello"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  // base class의 것PreTranslateMessage을 호출하는  return CFrameWindowImpl<CMyWindow>::PreTranslateMessage(pMsg);  }  // 아이들 처리  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLO, OnMenuHello)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 리버를 작성  CreateSimpleReBar();  // 툴바를 작성해 밴드에 추가  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  AddSimpleReBarBand(hWndToolBar);  // 상태 바를 작성  CreateSimpleStatusBar();    // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuHello(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello, ATL/WTL"));  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // hello.cpp  #include "stdafx.h"  #include "resource.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  // 코먼 컨트롤 및 리버 초기화  AtlInitCommonControls(ICC\_WIN95\_CLASSES | ICC\_COOL\_CLASSES);  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |

　우선 자원ID이ATL\_IDS\_IDLEMESSAGE라고 하는 캐릭터 라인 자원을 추가합니다. 이것은 아이들 시에 상태 바에 표시되는 캐릭터 라인입니다. 여기에서는 「레이디」로서 일어납니다.   
  
  
  
　다음에 CMyWindow클래스OnCreate()중에서 CreateSimpleStatusBar()를 호출해 상태 바를 작성합니다.   
  
덧붙여 CreateSimpleStatusBar()에는 2개 버젼이 있습니다. 양쪽 모두 3개의 인수를 지정할 수 있습니다만 한편은 제1인수에 아이돌시에 표시하는 캐릭터 라인을 지정하는 것으로 , 한편은 제1인수에 아이돌시에 표시하는 캐릭터 라인 자원의 것ID을 지정하는 것입니다. 후자는 모든 인수를 생략 가능합니다. 이번은 모든 인수를 생략 했으므로 후자가 사용됩니다. 이 경우 제1인수에는 디폴트 인수로서ATL\_IDS\_IDLEMESSAGE가 지정됩니다. 양버젼 모두 제2인수는 상태 바의 스타일입니다. 디폴트 인수는 다음의 스타일입니다.

|  |
| --- |
| // atlframe.h내  WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS | SBARS\_SIZEGRIP |

양버젼 모두 제3인수는 식별자입니다. 디폴트 인수는ATL\_IDW\_STATUS\_BAR입니다.   
  
어느 쪽의 버젼의 것CreateSimpleStatusBar()도 결과적으로 내부에서 ::CreateStatusWindow()를 호출하고 있어 그 호출에 의해 얻을 수 있던 상태 바의 핸들을 CFrameWindowImpl클래스의 base class인CFrameWindowImplBase 클래스의 것m\_hWndStatusBar이라고 하는HWND형태의 멤버 변수에 대입합니다.   
  
CFrameWindowImpl클래스의 base class인CFrameWindowImplBase클래스에는WM\_MENUSELECT메세지 핸들러가 준비되어 있습니다. 이 때문에 메뉴를 선택했을 때의WM\_MENUSELECT메세지가 체인처의CFrameWindowImpl클래스에 보내지면 자동적으로m\_hWndStatusBar로 식별된 상태 바에 메뉴 자원의[prompt]로 설정한 캐릭터 라인이 표시됩니다.  
  


▣ 프레임 윈도우 > UI갱신 핸들러 – 팝업 메뉴 아이템

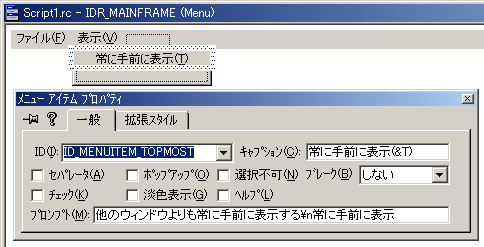
UI갱신 핸들러란 적절한 타이밍(아이들 시나 WM\_INITMENUPOPUP메세지가 보내져 왔을 때 등 ) 에 UI상태를 갱신하기 위한 구조입니다. 예를 들면 메뉴 아이템에 체크 마크를 넣거나 툴바 버튼을 사용 불가로 할 때 등에 사용합니다. UI갱신 핸들러는WTL의CUpdateUI클래스에 의해 실현됩니다.   
  
　여기에서는 전회의 상태 바를 추가한 「Hello, ATL/WTL」프로그램의 도구모음에 [항상 앞에 표시]라고 하는 메뉴 아이템을 추가합니다. 이 때 메뉴 아이템이 선택될 때마다 UI갱신 핸들러에 의해 메뉴 아이템의 체크 마크의 부가와 소거를 반복하도록 합니다.



|  |
| --- |
| // stdafx.h안  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // CFrameWindowImpl클래스를 사용하기 위해 |

|  |
| --- |
| // MainWindow.h안  // 메인 윈도우의 base class를CFrameWindowImpl로 변경  class CMyWindow : public CFrameWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler, public CUpdateUI<CMyWindow>  {  public:  // 윈도우 클래스명 , 공통 자원ID, 스타일 , 배경색을 등록  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("Hello"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  // UI갱신 핸들러 맵  BEGIN\_UPDATE\_UI\_MAP(CMyWindow)  UPDATE\_ELEMENT(ID\_MENUITEM\_TOPMOST, UPDUI\_MENUPOPUP)  END\_UPDATE\_UI\_MAP()  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  // base class의 것PreTranslateMessage을 호출하는  return CFrameWindowImpl<CMyWindow>::PreTranslateMessage(pMsg);  }  // 아이들 처리  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOPMOST, OnMenuTopmost)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLO, OnMenuHello)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMyWindow>) // CUpdateUI클래스에 체인  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 리버를 작성  CreateSimpleReBar();  // 툴바를 작성해 밴드에 추가  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  AddSimpleReBarBand(hWndToolBar);  // 상태 바를 작성  CreateSimpleStatusBar();  // [항상 앞에 표시]메뉴의 체크 마크 설정  UISetCheck(ID\_MENUITEM\_TOPMOST, false);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuTopmost(UINT uNotifyCode, int nID, HWND hWndCtl){  bool bTopmost = !(UIGetState(nID) & UPDUI\_CHECKED);  SetWindowPos(bTopmost ? HWND\_TOPMOST : HWND\_NOTOPMOST,  0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE | SWP\_SHOWWINDOW);  UISetCheck(nID, bTopmost);  }  void OnMenuHello(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello, ATL/WTL"));  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // hello.cpp내  #include "stdafx.h"  #include "resource.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  // 코먼 컨트롤 및 리버 초기화  AtlInitCommonControls(ICC\_WIN95\_CLASSES | ICC\_COOL\_CLASSES);  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |

　우선 도구모음의 톱 레벨에[표시]를 그 아래에[항상 앞에 표시]메뉴 아이템을 추가합니다. [항상 앞에 표시]메뉴 아이템에는ID\_MENUITEM\_TOPMOST이라고 하는 자원ID을 설정합니다.   
  
  
  
　CUpdateUI클래스는, 메인 윈도우인CMyWindow클래스의 base class로서 사용합니다. CMyWindow클래스에서는UI갱신 핸들러 맵을 준비해 갱신하는 메뉴 아이템의 자원ID과UI의 타입을 등록합니다. 덧붙여UI갱신 핸들러 맵은public나오지 않으면 안됩니다.

|  |
| --- |
| public:  // UI갱신 핸들러 맵  BEGIN\_UPDATE\_UI\_MAP(CMyWindow)  UPDATE\_ELEMENT(ID\_MENUITEM\_TOPMOST, UPDUI\_MENUPOPUP)  END\_UPDATE\_UI\_MAP() |

이 예에서는 pop-up 타입의 메뉴 아이템을 등록하므로 ,UI의 타입을UPDUI\_MENUPOPUP로 하고 있습니다. UI의 타입은 이것을 포함하고 다음과 같은 것이 준비되어 있습니다.

|  |  |
| --- | --- |
| UI타입 | 설명 |
| UPDUI\_MENUPOPUP | pop-up menu(도구모음하의 메뉴 아이템 포함한다 ) |
| UPDUI\_MENUBAR | 도구모음(톱 레벨의 메뉴 아이템) |
| UPDUI\_CHILDWINDOW | 자 윈도우 |
| UPDUI\_TOOLBAR | 툴바 |
| UPDUI\_STATUSBAR | 상태 바 |

　다음에 메세지 맵에 base class인CUpdateUI클래스에 체인을 추가합니다. 이것은 이번 예에서는 pop-up menu 아이템을 갱신하기 때문입니다. CUpdateUI클래스에 체인 하는 것에 의해 메인 윈도우에WM\_INITMENUPOPUP메세지가 보내져 올 때마다 CUpdateUI클래스에 메세지가 보내집니다. CUpdateUI클래스의 base class이려면CUpdateUIBaseWM\_INITMENUPOPUP메세지 핸들러가 준비되어 있어 그런데 pop-up menu 아이템 상태가 갱신됩니다.   
  
　다음에 [항상 앞에 표시]메뉴 아이템이 선택되었을 때의 커멘드 메세지 핸들러를 추가합니다. 메세지 맵에ID가ID\_MENUITEM\_TOPMOST의 커멘드 메세지용의 엔트리를 추가해 OnMenuTopmost()라고 하는 메세지 핸들러를 호출하도록(듯이) 합니다. 이 핸들러에서는 ,UIGetState()에 의해 현재의 체크 상태를 취득해 CUpdateUI클래스의 멤버 함수인UISetCheck()으로 체크 마크의 부가와 소거를 바꾸고 있습니다.   
  
여기서 주의하지 않으면 안 되는 것은 UISetCheck()을 호출한 직후에 pop-up menu 아이템의 체크 마크가 갱신되는 것이 아니라, 어디까지나 UISetCheck()를 호출한 후에WM\_INITMENUPOPUP메세지가 보내져 오고 나서 , UI갱신 핸들러에 의해 자동적으로 갱신된다고 하는 것입니다.

▣ 프레임 윈도우 > UI갱신 핸들러 – 툴바

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 전회는 pop-up menu 아이템의 갱신 핸들러를 추가했습니다만 이번은 거기에 더해 툴바의 갱신 핸들러를 추가합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl06-07/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // CFrameWindowImpl클래스를 사용하기 위해(때문에) |  |  | | --- | | // MainWindow.h  // 메인 윈도우의 base class를CFrameWindowImpl로 변경  class CMyWindow : public CFrameWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler, public CUpdateUI<CMyWindow>  {  public:  // 윈도우 클래스명 , 공통 자원ID, 스타일 , 배경색을 등록  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("Hello"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  // UI갱신 핸들러 맵  BEGIN\_UPDATE\_UI\_MAP(CMyWindow)  UPDATE\_ELEMENT(ID\_MENUITEM\_TOPMOST, UPDUI\_MENUPOPUP | UPDUI\_TOOLBAR)  END\_UPDATE\_UI\_MAP()  private:  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  // base class의 것PreTranslateMessage을 호출하는  return CFrameWindowImpl<CMyWindow>::PreTranslateMessage(pMsg);  }  // 아이들 처리  virtual BOOL OnIdle(){  UIUpdateToolBar();  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOPMOST, OnMenuTopmost)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLO, OnMenuHello)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMyWindow>) // CUpdateUI클래스에 체인  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 리버를 작성  CreateSimpleReBar();  // 툴바를 작성해 밴드에 추가  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  AddSimpleReBarBand(hWndToolBar);  UIAddToolBar(hWndToolBar);  // 상태 바를 작성  CreateSimpleStatusBar();  // [항상 앞에 표시]메뉴의 체크 마크 설정  UISetCheck(ID\_MENUITEM\_TOPMOST, false);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuTopmost(UINT uNotifyCode, int nID, HWND hWndCtl){  bool bTopmost = !(UIGetState(nID) & UPDUI\_CHECKED);  SetWindowPos(bTopmost ? HWND\_TOPMOST : HWND\_NOTOPMOST,  0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE | SWP\_SHOWWINDOW);  UISetCheck(nID, bTopmost);  }  void OnMenuHello(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello, ATL/WTL"));  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // hello.cpp내  #include "stdafx.h"  #include "resource.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  // 코먼 컨트롤 및 리버 초기화  AtlInitCommonControls(ICC\_WIN95\_CLASSES | ICC\_COOL\_CLASSES);  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　우선 툴바에 버튼을 하나 추가해 ID\_MENUITEM\_TOPMOST라고 하는 자원ID을 설정합니다. 이것은 벌써[항상 앞에 표시]메뉴 아이템용으로 작성했다ID입니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl06-07/02.png  　다음에 UI갱신 핸들러 맵에 갱신하는 툴바 버튼의 자원ID과UI의 타입을 등록합니다. 툴바 버튼의UI타입은UPDUI\_TOOLBAR입니다. 덧붙여 전회의[항상 앞에 표시]메뉴 아이템과 같은ID을 사용하고 있으므로 다음과 같이 추가합니다.   |  | | --- | | public:  // UI갱신 핸들러 맵  BEGIN\_UPDATE\_UI\_MAP(CMyWindow)  UPDATE\_ELEMENT(ID\_MENUITEM\_TOPMOST, UPDUI\_MENUPOPUP | UPDUI\_TOOLBAR)  END\_UPDATE\_UI\_MAP() |   　다음에 CMyWindow::OnIdle() 그리고 UIUpdateToolBar()를 호출합니다. 전회의 pop-up menu 아이템의 경우는WM\_INITMENUPOPUP메세지가 보내진 타이밍에 갱신되었습니다만 그 이외의 것UI(이번 예에서는 툴바) 의 경우는 아이돌시에 갱신용 멤버 함수를 호출해 갱신합니다. 이 때문에 CUpdateUI클래스는 툴바도 포함해 다음과 같은 갱신용 멤버 함수를 준비해 있습니다.   |  |  | | --- | --- | | 멤버 함수명 | 설명 | | UIUpdateMenuBar | 도구모음(톱 레벨의 메뉴 아이템) 을 갱신 | | UIUpdateChildWindows | 자 윈도우를 갱신 | | UIUpdateToolBar | 툴바를 갱신 | | UIUpdateStatusBar | 상태 바를 갱신 |   　다음에 CMyWindow::OnCreate()그리고UIAddToolBar()를 호출합니다. UIAddToolBar()의 인수에는 갱신하는 툴바의 핸들을 지정합니다. UI갱신 핸들러는 여기서 지정된 핸들로 식별되는 툴바에 대해서 갱신 처리를 실시합니다. CUpdateUI클래스는 툴바도 포함해 다음과 같은 핸들 추가용 멤버 함수를 준비해 있습니다.   |  |  | | --- | --- | | 멤버 함수명 | 설명 | | UIAddMenuBar | 도구모음(톱 레벨의 메뉴 아이템) 의 핸들을 추가 | | UIAddChildWindowContainer | 자 윈도우의 핸들을 추가 | | UIAddToolBar | 툴바의 핸들을 추가 | | UIAddStatusBar | 상태 바의 핸들을 추가 |   　이렇게 해 CMyWindow::OnMenuTopmost()의UISetCheck()에 의해 체크 마크의 부가와 소거가 교체될 때마다 툴바의 버튼도 밀린 상태와 밀리지 않은 상태가 바뀝니다.   여기서 주의하지 않으면 안 되는 것은 UISetCheck()을 호출한 직후에 툴바의 버튼 상태가 갱신되는 것이 아니라 어디까지나 UISetCheck()를 호출한 후 의 아이들시에 UI갱신 핸들러에 의해 자동적으로 갱신된다고 하는 것입니다. |

▣ 프레임 윈도우 > 뷰 윈도우

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 뷰 윈도우란 프레임 윈도우의 클라이언트 영역을 가리도록 해 표시되는 아이 윈도우입니다. 여기에서는 지금까지 작성해 온 프레임 윈도우에 뷰윈드우를 추가해 그 뷰윈드우에 「Hello, ATL/WTL」이라고 하는 캐릭터 라인을 묘화 합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl06-08/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // CFrameWindowImpl클래스를 사용하기 위해(때문에) |  |  | | --- | | // View.h  class CView : public CWindowImpl<CView>  {  public:  // 메세지 필터 처리  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CView)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON,  pt.x, pt.y,  GetParent() // 친윈도우의 핸들  );  }else{  SetMsgHandled(false);  }  }  }; |  |  | | --- | | // MainWindow.h내  // 메인 윈도우의 base class를CFrameWindowImpl로 변경  class CMyWindow : public CFrameWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler, public CUpdateUI<CMyWindow>  {  public:  // 윈도우 클래스명 , 공통 자원ID을 등록  DECLARE\_FRAME\_WND\_CLASS(\_T("Hello"), IDR\_MAINFRAME)  // UI갱신 핸들러 맵  BEGIN\_UPDATE\_UI\_MAP(CMyWindow)  UPDATE\_ELEMENT(ID\_MENUITEM\_TOPMOST, UPDUI\_MENUPOPUP | UPDUI\_TOOLBAR)  END\_UPDATE\_UI\_MAP()  private:  CView m\_view;  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  // base class의 것PreTranslateMessage을 호출하는  if(CFrameWindowImpl<CMyWindow>::PreTranslateMessage(pMsg))  return TRUE;  // 뷰윈드우크라스의 것PreTranslateMessage을 호출하는  return m\_view.PreTranslateMessage(pMsg);  }  // 아이들 처리  virtual BOOL OnIdle(){  UIUpdateToolBar();  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOPMOST, OnMenuTopmost)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLO, OnMenuHello)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMyWindow>) // CUpdateUI클래스에 체인  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 리버를 작성  CreateSimpleReBar();  // 툴바를 작성해 밴드에 추가  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  AddSimpleReBarBand(hWndToolBar);  UIAddToolBar(hWndToolBar);  // 상태 바를 작성  CreateSimpleStatusBar();  // 뷰윈드우를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN,  WS\_EX\_CLIENTEDGE);  // [항상 앞에 표시]메뉴의 체크 마크 설정  UISetCheck(ID\_MENUITEM\_TOPMOST, false);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuTopmost(UINT uNotifyCode, int nID, HWND hWndCtl){  bool bTopmost = !(UIGetState(nID) & UPDUI\_CHECKED);  SetWindowPos(bTopmost ? HWND\_TOPMOST : HWND\_NOTOPMOST,  0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE | SWP\_SHOWWINDOW);  UISetCheck(nID, bTopmost);  }  void OnMenuHello(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello, ATL/WTL"));  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // hello.cpp  #include "stdafx.h"  #include "resource.h"  #include "View.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  // 코먼 컨트롤 및 리버 초기화  AtlInitCommonControls(ICC\_WIN95\_CLASSES | ICC\_COOL\_CLASSES);  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　우선, 뷰윈드우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에View.h라고 하는 헤더 파일을 추가해 거기에 CView라고 하는 클래스를 정의하고 있습니다. CView클래스는CWindowImpl클래스로부터 파생하는 통상의 윈도우입니다. CView클래스내에서는 , 독자적인 메세지 필터를 준비해 메세지 맵으로 WM\_PAINT메시지 와 WM\_CONTEXTMENU메세지를 매핑 하고 있습니다. 이것들 두 개의 메세지는 지금까지 프레임 윈도우측에서 처리하고 있었습니다만 뷰윈드우에 의해 프레임 윈도우의 클라이언트 영역이 숨어 버리기 때문에 뷰 윈도우 측에서 처리합니다. 따라서 프레임 윈도우인CMyWindow클래스로부터는 이것들 두 개의 메세지 맵 엔트리 및 메세지 핸들러를 삭제하고 있습니다.   덧붙여WM\_CONTEXTMENU메세지 핸들러로 호출하고 있는 TrackPopupMenu()의 제4인수는 부모 윈도우의 핸들로 변경합니다.   　CMyWindow클래스내에서는 , 우선 프레임 윈도우 클래스명 등록 매크로를 DECLARE\_FRAME\_WND\_CLASS\_EX로부터 DECLARE\_FRAME\_WND\_CLASS로 변경하고 있습니다. 이것은 뷰윈드우를 사용하는 것에 의해 프레임 윈도우의 클라이언트 영역에 직접 「Hello, ATL/WTL」이라고 하는 캐릭터 라인을 묘화 하지 않게 되기 때문에 CS\_HREDRAW | CS\_VREDRAW 라고 하는 스타일을 지정할 필요 없게 되기 때문입니다.   　다음에 뷰윈드우인CView클래스의 인스턴스를 CMyWindow클래스의 멤버 변수로서 선언합니다. CMyWindow::OnCreate()그럼 이 뷰 윈도우를 작성해 CFrameWindowImpl클래스의 base class인 CFrameWindowImplBase 클래스의 m\_hWndClient이라고 하는HWND 형태의 멤버 변수에 대입합니다.   CFrameWindowImpl클래스에는WM\_SIZE메세지 핸들러가 준비되어 있습니다. 이 때문에 메인 윈도우의 사이즈를 변경했을 때의WM\_SIZE메세지가 체인처의CFrameWindowImpl클래스에 보내지면 CFrameWindowImpl클래스의 WM\_SIZE메세지 핸들러가 불려 갑니다. 이 핸들러에서는UpdateLayout()이라고 하는 함수를 호출하고 있어 이 함수는 각 m\_hWndClient에서 식별되는 뷰윈드우를 자동적으로 프레임 윈도우의 사이즈에 맞춥니다.   그리고 CMyWindow클래스의 메세지 필터에서는 CView클래스의 독자 메세지 필터 함수를 호출하고 있습니다. 이것에 의해 뷰윈드우에도 메세지 필터링의 기회를 주고 있습니다.   　hello.cpp파일내에서는 ,MainWindow.h헤더를 인클루드 하기 전에View.h헤더를 인클루드 합니다. |

▣ 프레임 윈도우 > 커맨드 체인

　통상 도구모음이나 툴바로부터의WM\_COMMAND메세지는 메인 윈도우의 메세지 맵이 처리합니다. 그러나 특정의 클래스나 뷰 윈도우의 메세지 맵으로 처리하고 싶은 경우도 있습니다. 그러한 때는 WTL의 커멘드 체인 매크로를 사용합니다. ATL도 몇개의 체인 매크로를 준비해 있습니다만 커멘드 체인 매크로는WM\_COMMAND메세지 전용의 체인 매크로입니다.   
  
WTL(은)는 다음과 같은 커멘드 체인 매크로를 준비해 있습니다.

CHAIN\_COMMANDS(클래스명)   
WM\_COMMAND메세지를 , 지정된 클래스의 메세지 맵에 체인 합니다.

CHAIN\_COMMANDS\_MEMBER(인스턴스명)   
WM\_COMMAND메세지를 , 지정된 인스턴스의 메세지 맵에 체인 합니다.

CHAIN\_COMMANDS\_ALT(클래스명, 대체 메세지 맵 번호)   
WM\_COMMAND메세지를 , 지정된 클래스의 대체 메세지 맵에 체인 합니다.

CHAIN\_COMMANDS\_ALT\_MEMBER(인스턴스명, 대체 메세지 맵 번호)   
WM\_COMMAND메세지를 , 지정된 인스턴스의 대체 메세지 맵에 체인 합니다.

CHAIN\_CLIENT\_COMMANDS()   
WM\_COMMAND메세지를 ,m\_hWndClient그리고 식별되는 뷰윈드우의 메세지 맵에 체인 합니다.

　다음에 나타내는 것은 전회 뷰 윈도우를을 추가한 「Hello, ATL/WTL」프로그램에 커멘드 체인을 추가해 뷰 윈도우 측에서 ID가ID\_MENUITEM\_HELLO의 커멘드 메세지를 처리하는 예입니다.

|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // CFrameWindowImpl클래스를 사용하기 위해(때문에) |

|  |
| --- |
| // View.h  class CView : public CWindowImpl<CView>  {  public:  // 메세지 필터 처리  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CView)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLO, OnMenuHello)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON,  pt.x, pt.y,  GetParent() // 친윈도우의 핸들  );  }else{  SetMsgHandled(false);  }  }  void OnMenuHello(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello, ATL/WTL"));  }  }; |

|  |
| --- |
| // MainWindow.h  // 메인 윈도우의 base class를CFrameWindowImpl로 변경  class CMyWindow : public CFrameWindowImpl<CMyWindow>,  public CMessageFilter, public CIdleHandler, public CUpdateUI<CMyWindow>  {  public:  // 윈도우 클래스명 , 공통 자원ID을 등록  DECLARE\_FRAME\_WND\_CLASS(\_T("Hello"), IDR\_MAINFRAME)  // UI갱신 핸들러 맵  BEGIN\_UPDATE\_UI\_MAP(CMyWindow)  UPDATE\_ELEMENT(ID\_MENUITEM\_TOPMOST, UPDUI\_MENUPOPUP | UPDUI\_TOOLBAR)  END\_UPDATE\_UI\_MAP()  private:  CView m\_view;  // 메세지 필터 처리  virtual BOOL PreTranslateMessage(MSG\* pMsg){  // base class의 것PreTranslateMessage을 호출하는  if(CFrameWindowImpl<CMyWindow>::PreTranslateMessage(pMsg))  return TRUE;  // 뷰윈드우크라스의 것PreTranslateMessage을 호출하는  return m\_view.PreTranslateMessage(pMsg);  }  // 아이돌 처리  virtual BOOL OnIdle(){  UIUpdateToolBar();  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOPMOST, OnMenuTopmost)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMyWindow>) // CUpdateUI클래스에 체인  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  CHAIN\_CLIENT\_COMMANDS() // 뷰윈드우에 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 리버를 작성  CreateSimpleReBar();  // 툴바를 작성해 밴드에 추가  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  AddSimpleReBarBand(hWndToolBar);  UIAddToolBar(hWndToolBar);  // 상태 바를 작성  CreateSimpleStatusBar();  // 뷰윈드우를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN,  WS\_EX\_CLIENTEDGE);  // [항상 앞에 표시]메뉴의 체크 마크 설정  UISetCheck(ID\_MENUITEM\_TOPMOST, false);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuTopmost(UINT uNotifyCode, int nID, HWND hWndCtl){  bool bTopmost = !(UIGetState(nID) & UPDUI\_CHECKED);  SetWindowPos(bTopmost ? HWND\_TOPMOST : HWND\_NOTOPMOST,  0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE | SWP\_SHOWWINDOW);  UISetCheck(nID, bTopmost);  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

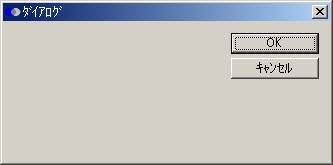
|  |
| --- |
| // hello.cpp  #include "stdafx.h"  #include "resource.h"  #include "View.h"  #include "MainWindow.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  // 코먼 컨트롤 및 리버 초기화  AtlInitCommonControls(ICC\_WIN95\_CLASSES | ICC\_COOL\_CLASSES);  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  // 독자 윈도우를 작성  CMyWindow wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |

　우선 ID이ID\_MENUITEM\_HELLO의 커멘드 메세지용 메세지 맵 엔트리와 메세지 핸들러를 CMyWindow클래스에서CView클래스에 이동합니다. 그리고 CMyWindow클래스의 메세지 맵에 CHAIN\_CLIENT\_COMMANDS() 커멘드 체인 매크로를 추가합니다.   
  
이것에 의해 메인 윈도우인CMyWindow클래스에서 처리되지 않는 ID\_MENUITEM\_HELLO커멘드 메세지는 뷰 윈도우인 CView클래스에 보내져 CView클래스의 OnMenuHello()이 불려 가게 됩니다.   
  
덧붙여 위의 예에서는CHAIN\_CLIENT\_COMMANDS()매크로를 사용했습니다만 CHAIN\_COMMANDS\_MEMBER()매크로에서도 같은 것이 생깁니다. 그 경우는 뷰 윈도우 클래스의 인스턴스를 지정합니다.

|  |
| --- |
| CView m\_view; // 뷰윈드우의 인스턴스  ...  ...  // CMyWindow클래스의 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMyWindow)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOPMOST, OnMenuTopmost)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMyWindow>) // CUpdateUI클래스에 체인  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMyWindow>) // CFrameWindowImpl클래스에 체인  CHAIN\_COMMANDS\_MEMBER(m\_view) // 뷰윈드우에 체인  END\_MSG\_MAP() |

▣ 다이얼로그 > 모달 다이얼로그

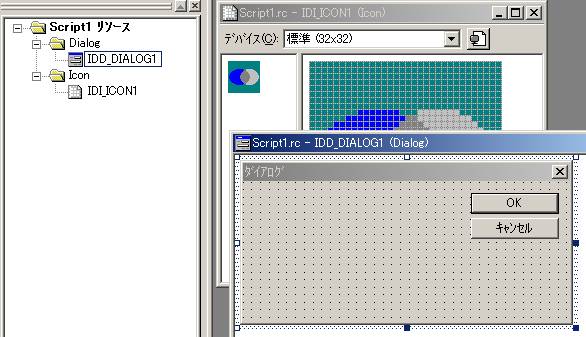
윈도우를 작성하기 위해서는CWindowImpl클래스를 사용했습니다만, 다이얼로그를 작성하기 위해서는CDialogImpl클래스를 사용합니다.   
  
　이하에 나타내는 것은 모덜 다이얼로그를 표시하기만 하면 되는 간단한 프로그램의 원시 코드입니다. 이 프로그램에서는  CDialogImpl클래스로부터 상속 클래스 CMainDlg(이것이 메인 윈도우가 됩니다 ) 를 만들어 CMainDlg 클래스내에서 WM\_INITDIALOG 메세지에의 응답과 IDOK,IDCANCEL라고 하는 ID를 가지는 WM\_COMMAND 메세지에의 응답을 정의하고 있습니다. 덧붙여 이 프로그램은 Win32Application 프로젝트로 빌드 합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>    #include <atlcrack.h>  #include <atlmisc.h> |

|  |
| --- |
| // MainDlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:      enum { IDD = IDD\_DIALOG1 };        // 메세지 맵      BEGIN\_MSG\_MAP\_EX(CMainDlg)          MSG\_WM\_INITDIALOG(OnInitDialog)          COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)          COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)      END\_MSG\_MAP()        LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){          // 스크린의 중앙에 배치          CenterWindow();            // 큰 아이콘 설정          HICON hIcon = AtlLoadIconImage(IDI\_ICON\_MAIN, LR\_DEFAULTCOLOR,              ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));          SetIcon(hIcon, TRUE);          // 작은 아이콘 설정          HICON hIconSmall = AtlLoadIconImage(IDI\_ICON\_MAIN, LR\_DEFAULTCOLOR,              ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));          SetIcon(hIconSmall, FALSE);            return TRUE;      }        void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){          EndDialog(nID);      }        void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){          EndDialog(nID);      }  }; |

|  |
| --- |
| // modal.cpp #include "stdafx.h"    #include "resource.h"    #include "MainDlg.h"    CAppModule \_Module;    int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {      \_Module.Init(NULL, hInstance);        CMainDlg dlgMain;      int nRet = dlgMain.DoModal();        \_Module.Term();        return nRet;  } |

　우선은 프로젝트에 다이얼로그 자원과 아이콘 자원을 추가해 자원ID으로서 각각 IDD\_DIALOG1IDI\_ICON1 를 설정합니다.   
  
  
  
　다음에  윈도우 때와 같이 CAppModule 클래스의 인스턴스를 글로벌 하게 선언해  \_tWinMain()의 최초와 최후로 초기화와 뒤처리를 하고 있습니다. 모덜 다이얼로그를 작성하려면 DoModal()를 호출합니다.   
  
　CMainDlg 클래스는 CDialogImpl 클래스로부터 파생하고 있습니다만  CDialogImpl의 제1템플릿 인수에도 CMainDlg라는 이름을 건네줍니다 ( 제2템플릿 인수는 생략 가능합니다. 여기에서는 생략 하고 있습니다. ). CMainDlg클래스내에서는 public 선언의  enum에 의해 다이얼로그 자원 ID를 정의해  메세지 맵에 의해 메세지와 그에 대한 핸들러를 묶고 있습니다.   
  
이 프로그램에서는 자원 ID가 IDOK 또는 IDCANCEL 의 버튼을 누르면 프로그램을 종료하도록 하고 있습니다. 이번 예에서는 모덜 다이얼로그가 메인 윈도우가 되므로  버튼을 눌렀을 때에 EndDialog()를 호출해 다이얼로그를 다물고 있습니다.

▣ 다이얼로그 > 모달리스 다이얼로그

|  |  |  |  |
| --- | --- | --- | --- |
| 모달리스 다이얼로그를 작성하는 경우도 모덜 다이얼로그와 같게 CDialogImpl클래스를 사용합니다.   　이하에 나타내는 것은 모달리스 다이얼로그를 표시하기만 하면 되는 간단한 프로그램의 원시 코드입니다. 이 프로그램에서는 모덜 다이얼로그와 같이 CDialogImpl 클래스로부터 상속 클래스 CMainDlg( 이것이 메인 윈도우가 됩니다 ) 를 만들어 CMainDlg 클래스내에서 WM\_INITDIALOG 메세지에의 응답과 IDOK, IDCANCEL라고 하는 ID를 가지는 WM\_COMMAND 메세지에의 응답을 정의하고 있습니다. 덧붙여 이 프로그램은 Win32 Application 프로젝트로 빌드 합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl07-02/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h> |  |  | | --- | | // MainDlg.h  class CMainDlg : public CDialogImpl<CMainDlg>,  public CMessageFilter, public CIdleHandler  {  public:  enum { IDD = IDD\_DIALOG1 };  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return IsDialogMessage(pMsg);  }  virtual BOOL OnIdle(){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDI\_ICON\_MAIN, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDI\_ICON\_MAIN, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  DestroyWindow();  ::PostQuitMessage(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  DestroyWindow();  ::PostQuitMessage(nID);  }  }; |  |  | | --- | | // modaless.cpp  #include "stdafx.h"  #include "resource.h"  #include "MainDlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainDlg dlgMain;  dlgMain.Create(NULL);  dlgMain.ShowWindow(nCmdShow);  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　윈도우때와 같이 CAppModule 클래스의 인스턴스를 글로벌하게 선언해 \_tWinMain()의 최초와 최후로 초기화와 뒤처리를 하고 있습니다. 모달리스 다이얼로그를 작성하려면 Create()를 호출합니다.   　CMainDlg 클래스는 CDialogImpl 클래스로부터 파생하고 있습니다만 CDialogImpl의 제1템플릿 인수에도 CMainDlg 라는 이름을 건네줍니다 ( 제2템플릿 인수는 생략 가능합니다. 여기에서는 생략 하고 있습니다. ). CMainDlg 클래스내에서는 public 선언의 enum에 의해 다이얼로그 자원 ID을 정의해 , 메세지 맵에 의해 메세지와 그에 대한 핸들러를 묶고 있습니다.   이 프로그램에서는 자원 ID가 IDOK 또는 IDCANCEL의 버튼을 누르면 프로그램을 종료하도록 하고 있습니다. 이번 예에서는 모달리스 다이얼로그가 메인 윈도우가 되므로 버튼을 눌렀을 때에 DestroyWindow()를 호출해 윈도우를 파기해 PostQuitMessage()를 호출해 메시지 루프를 종료시키고 있습니다.   　덧붙여 모달리스 다이얼로그는 모덜 다이얼로그와 달리CMessageLoop 클래스에서 메시지 루프를 관리할 수 있기 때문에 CMessageFilter 클래스나 CIdleHandler 클래스를 사용해 메세지 필터나 아이들 핸들러를 이용할 수 있습니다. 위의 예에서는 메세지 필터로 IsDialogMessage()를 호출하고 있습니다. |

▣ 다이얼로그 > UI갱신 핸들러 – 다이얼로그의 팝업 메뉴 아이템

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 모덜 다이얼로그에서도 모달리스 다이얼로그에서도  pop-up menu 아이템의 UI갱신 핸들러를 사용할 수 있습니다.   　아래에 나타내는 것은  지금까지 작성한 다이얼로그에 pop-up menu 아이템의 UI갱신 핸들러를 추가하는 순서입니다. 여기에서는  다이얼로그에 도구모음을 추가해 [항상 앞에 표시]라고 하는 메뉴 아이템을 작성해 메뉴 아이템이 선택될 때마다 UI 갱신 핸들러에 의해 메뉴 아이템의 체크 마크의 부가와 소거를 반복하도록 합니다. 덧붙여 여기 가리키는 순서는 모덜 다이얼로그 , 모달리스 다이얼로그의 어느 쪽에도 들어 맞읍니다.  http://jacking75.cafe24.com/WTL/33.files/image001.jpg  　우선은 프로젝트에 IDR\_MENU\_MAIN라고 하는ID의 메뉴 자원을 추가해  [항상 앞에 표시]라고 하는 메뉴 아이템을 작성해 거기에ID\_MENUITEM\_TOPMOST라고 하는ID를 설정합니다. 다음에  다이얼로그의 프로퍼티를 열어  메뉴로서IDR\_MENU\_MAIN를 지정합니다. 이것으로 다이얼로그에 메뉴가 추가됩니다.   http://jacking75.cafe24.com/WTL/33.files/image002.jpg  　다음에 UI갱신 핸들러를 사용하기 위해서는CUpdateUI클래스가 필요하게 되기 때문에  atlframe.h헤더를 인클루드 합니다.   |  | | --- | | // stdafx.h내  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> |   　다음에 CUpdateUI  클래스를 다이얼로그 클래스인 CMainDlg클래스의 base class로 합니다.   |  | | --- | | class CMainDlg : public CDialogImpl<CMainDlg>, public CUpdateUI<CMainDlg> |   　CMainDlg 클래스 내에서는 UI갱신 핸들러 맵에ID\_MENUITEM\_TOPMOST 용무의 엔트리를 추가합니다. 한층 더 메세지 맵에는ID\_MENUITEM\_TOPMOST 용무의 커멘드 핸들러 엔트리와  CUpdateUI클래스에의 체인을 추가합니다.   |  | | --- | | class CMainDlg : public CDialogImpl<CMainDlg>, public CUpdateUI<CMainDlg>  {  public:      enum { IDD = IDD\_DIALOG1 };        // UI갱신 핸들러 맵      BEGIN\_UPDATE\_UI\_MAP(CMainDlg)          UPDATE\_ELEMENT(ID\_MENUITEM\_TOPMOST, UPDUI\_MENUPOPUP)      END\_UPDATE\_UI\_MAP()        // 메세지 맵      BEGIN\_MSG\_MAP\_EX(CMainDlg)          MSG\_WM\_INITDIALOG(OnInitDialog)          COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOPMOST, OnMenuTopmost)  // [항상 앞에 표시]          ...          ...          CHAIN\_MSG\_MAP(CUpdateUI<CMainDlg>)  // CUpdateUI클래스에의 체인      END\_MSG\_MAP()        ... |   　다음에 CMainDlg 클래스에ID\_MENUITEM\_TOPMOST의 커멘드 메세지 핸들러를 추가합니다.   |  | | --- | | class CMainDlg : public CDialogImpl<CMainDlg>, public CUpdateUI<CMainDlg>  {      ...      ...        void OnMenuTopmost(UINT uNotifyCode, int nID, HWND hWndCtl){          bool bTopmost = !(UIGetState(nID) & UPDUI\_CHECKED);          SetWindowPos(bTopmost ? HWND\_TOPMOST : HWND\_NOTOPMOST,              0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE | SWP\_SHOWWINDOW);          UISetCheck(nID, bTopmost);      }       ... |   이것으로  pop-up menu 아이템용의 UI갱신 핸들러가 움직이게 됩니다. 우선  다이얼로그에 메뉴를 추가하는 것으로  메뉴가 표시될  때 WM\_INITMENUPOPUP 메세지가 다이얼로그에 보내져 오게 됩니다.  CMainDlg 클래스의 메시지  맵에서는 WM\_INITMENUPOPUP 메세지를 처리하지 않기 때문에  이 메세지는 체인처의 CUpdateUI클래스에  보내집니다.  CUpdateUI 클래스의 base class 이려면 CUpdateUIBase  WM\_INITMENUPOPUP 메세지 핸들러가 준비되어 있어 pop-up menu 아이템이 갱신됩니다. |

▣ 다이얼로그 > UI갱신 핸들러 – 자식 윈도우

|  |  |  |  |
| --- | --- | --- | --- |
| 전회의 pop-up menu 아이템용의UI갱신 핸들러는  모덜 다이얼로그에서도 모달리스 다이얼로그에서도 유효했습니다만  pop-up menu 아이템 이외 (도구모음 , 툴바 , 상태 바 , 아이 윈도우) 의 UI갱신 핸들러는 모달리스 다이얼로그로 밖에 사용할 수 없습니다. 왜냐하면  pop-up menu 아이템 이외의 UI갱신 핸들러는 기본적으로 아이들  핸들러로 불려 갑니다만  모덜 다이얼로그에는 아이들 핸들러가 없기 때문입니다.   　여기에서는  전회 pop-up menu 아이템용의UI갱신 핸들러를 추가한 모달리스 다이얼로그에  자식 윈도우용의 UI갱신 핸들러를 추가합니다. 이 때  메뉴의[항상 앞에 표시]의 온/오프에 의해  다이얼로그상의 [캔슬] 버튼을 사용가능/사용 불가가 되도록 해 보겠습니다. 이와 같이  자식 윈도우용의 UI갱신 핸들러는 주로  다이얼로그를  「컨테이너」로서  다이얼로그상의 컨트롤 상태를 갱신할 경우에 사용합니다.  http://jacking75.cafe24.com/WTL/34.files/image001.jpg    http://jacking75.cafe24.com/WTL/34.files/image002.jpg     |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>    #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> |      |  | | --- | | // MainDlg.h  class CMainDlg : public CDialogImpl<CMainDlg>,      public CMessageFilter, public CIdleHandler, public CUpdateUI<CMainDlg>  {  public:      enum { IDD = IDD\_DIALOG1 };        virtual BOOL PreTranslateMessage(MSG\* pMsg){          return IsDialogMessage(pMsg);      }        virtual BOOL OnIdle(){          UIUpdateChildWindows();          return FALSE;      }        // UI갱신 핸들러 맵      BEGIN\_UPDATE\_UI\_MAP(CMainDlg)          UPDATE\_ELEMENT(ID\_MENUITEM\_TOPMOST, UPDUI\_MENUPOPUP)          UPDATE\_ELEMENT(IDCANCEL, UPDUI\_CHILDWINDOW)      END\_UPDATE\_UI\_MAP()        // 메세지 맵      BEGIN\_MSG\_MAP\_EX(CMainDlg)          MSG\_WM\_INITDIALOG(OnInitDialog)          COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOPMOST, OnMenuTopmost)          COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)          COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)          CHAIN\_MSG\_MAP(CUpdateUI<CMainDlg>)      END\_MSG\_MAP()        LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){          // 스크린의 중앙에 배치          CenterWindow();            // 큰 아이콘 설정          HICON hIcon = AtlLoadIconImage(IDI\_ICON\_MAIN, LR\_DEFAULTCOLOR,              ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));          SetIcon(hIcon, TRUE);            // 작은 아이콘 설정          HICON hIconSmall = AtlLoadIconImage(IDI\_ICON\_MAIN, LR\_DEFAULTCOLOR,              ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));          SetIcon(hIconSmall, FALSE);            // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가          CMessageLoop\* pLoop = \_Module.GetMessageLoop();          pLoop->AddMessageFilter(this);          pLoop->AddIdleHandler(this);            // 다이얼로그 핸들을 아이 윈도우용의UI갱신 핸들러에 추가          UIAddChildWindowContainer(m\_hWnd);            // [캔슬]버튼의 사용 가부 설정          UIEnable(IDCANCEL, FALSE);            // [항상 앞에 표시]메뉴의 체크 마크 설정          UISetCheck(ID\_MENUITEM\_TOPMOST, false);            return TRUE;      }        void OnMenuTopmost(UINT uNotifyCode, int nID, HWND hWndCtl){          bool bTopmost = !(UIGetState(nID) & UPDUI\_CHECKED);          SetWindowPos(bTopmost ? HWND\_TOPMOST : HWND\_NOTOPMOST,              0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE | SWP\_SHOWWINDOW);          UISetCheck(nID, bTopmost);            // [캔슬]버튼의 사용 가부 설정          UIEnable(IDCANCEL, bTopmost);      }        void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){          DestroyWindow();          ::PostQuitMessage(nID);      }        void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){          DestroyWindow();          ::PostQuitMessage(nID);      }  }; |      |  | | --- | | // modaless.cpp  #include "stdafx.h"    #include "resource.h"    #include "MainDlg.h"    CAppModule \_Module;    int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {      \_Module.Init(NULL, hInstance);        CMessageLoop theLoop;      \_Module.AddMessageLoop(&theLoop);        CMainDlg dlgMain;      dlgMain.Create(NULL);      dlgMain.ShowWindow(nCmdShow);        int nRet = theLoop.Run();        \_Module.RemoveMessageLoop();        \_Module.Term();        return nRet;  } |   　자식 윈도우용의UI갱신 핸들러를 사용하려면 우선  아이들 핸들러에 자식 윈도우용의 UI갱신  핸들러인 UIUpdateChildWindows()를 추가합니다.   　다음에 UI갱신 핸들러 맵으로 상태를 갱신하는 컨트롤의 자원 ID와 UI타입을 등록합니다. 이번  예에서는 [캔슬] 버튼을 자식 윈도우용 UI갱신 핸들러로 갱신하기 때문에 자원ID 에는IDCANCEL, UI 타입에는 UPDUI\_CHILDWINDOW를 지정합니다.   　OnInitDialog()안에서는 컨테이너가 되는 다이얼로그의 핸들을 UIAddChildWindowContainer()에 의해 UI 갱신 핸들러에 추가합니다.  UI 갱신 핸들러에서는  여기서 지정된 다이얼로그상의 컨트롤에 대해서 갱신 처리를 실시합니다.   　마지막으로 OnMenuTopmost()안에서 UIEnable()를 사용하는 것에 의해  [캔슬] 버튼 상태를 바꾸고 있습니다.   여기서 주의하지 않으면 안 되는 것은 UIEnable()를 호출한 직후에  [캔슬] 버튼 상태가 갱신되는 것이 아니라 어디까지나 UIEnable()를 호출한 후 의  아이들 시에  UI갱신 핸들러에 의해 자동적으로 갱신된다고 하는 것입니다. |

▣ 다이얼로그 > 키보드 액셀레이터

|  |  |  |  |
| --- | --- | --- | --- |
| 프레임 윈도우의 경우는 가속기 테이블 자원을 매크로에 의해 설정할 수 있었습니다. 그리고 프레임 윈도우를 서포트하는CFrameWindowImpl클래스는 키보드 가속기를 유효하게하기 위한 기능을 갖추고 있었습니다.   　그런데 다이얼로그를 서포트하는CDialogImpl 클래스는 그러한 매크로나 기능을 갖추고 있지 않습니다. 그런데 이번은 전회 윈도우용의 UI갱신 핸들러를 추가한 모달리스 다이얼로그에 키보드 가속기를 추가하는 방법을 나타냅니다. 덧붙여 여기에서는 벌써 추가한 메뉴의 [항상 앞에 표시]에 [Ctrl+T] 라고 하는 가속기 키를 할당합니다.   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> |  |  | | --- | | // MainDlg.h  class CMainDlg : public CDialogImpl<CMainDlg>,  public CMessageFilter, public CIdleHandler, public CUpdateUI<CMainDlg>  {  public:  enum { IDD = IDD\_DIALOG1 };  HACCEL m\_hAccelTable; // 가속기 테이블의 핸들  // constructor  CMainDlg() : m\_hAccelTable(NULL)  {}  virtual BOOL PreTranslateMessage(MSG\* pMsg){  // 가속기  if(m\_hAccelTable != NULL){  if(::TranslateAccelerator(m\_hWnd, m\_hAccelTable, pMsg))  return TRUE;  }  return IsDialogMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateChildWindows();  return FALSE;  }  // UI갱신 핸들러 맵  BEGIN\_UPDATE\_UI\_MAP(CMyWindow)  UPDATE\_ELEMENT(ID\_MENUITEM\_TOPMOST, UPDUI\_MENUPOPUP)  UPDATE\_ELEMENT(IDCANCEL, UPDUI\_CHILDWINDOW)  END\_UPDATE\_UI\_MAP()  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOPMOST, OnMenuTopmost)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  CHAIN\_MSG\_MAP(CUpdateUI<CMainDlg>)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDI\_ICON\_MAIN, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDI\_ICON\_MAIN, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  // 다이얼로그 핸들을 아이 윈도우용의UI갱신 핸들러에 추가  UIAddChildWindowContainer(m\_hWnd);    // [캔슬]버튼의 사용 가부 설정  UIEnable(IDCANCEL, FALSE);    // [항상 앞에 표시]메뉴의 체크 마크 설정  UISetCheck(ID\_MENUITEM\_TOPMOST, false);  // 가속기 테이블을 로드  m\_hAccelTable = AtlLoadAccelerators(IDR\_ACCELERATOR\_MAIN);  return TRUE;  }  void OnMenuTopmost(UINT uNotifyCode, int nID, HWND hWndCtl){  bool bTopmost = !(UIGetState(nID) & UPDUI\_CHECKED);  SetWindowPos(bTopmost ? HWND\_TOPMOST : HWND\_NOTOPMOST,  0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE | SWP\_SHOWWINDOW);  UISetCheck(nID, bTopmost);  // [캔슬]버튼의 사용 가부 설정  UIEnable(IDCANCEL, bTopmost);  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  DestroyWindow();  ::PostQuitMessage(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  DestroyWindow();  ::PostQuitMessage(nID);  }  }; |  |  | | --- | | // modaless.cpp  #include "stdafx.h"  #include "resource.h"  #include "MainDlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  \_Module.Init(NULL, hInstance);  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainDlg dlgMain;  dlgMain.Create(NULL);  dlgMain.ShowWindow(nCmdShow);  int nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  \_Module.Term();  return nRet;  } |   　우선은 프로젝트에 가속기 테이블 자원을 추가해 자원 ID으로서 IDR\_ACCELERATOR\_MAIN을 설정합니다. 그리고 아이템으로서ID\_MENUITEM\_TOPMOST라고 하는 ID 를 추가해 Ctrl＋T 라고 하는 키를 등록합니다. 덧붙여ID\_MENUITEM\_TOPMOST라고 하는 ID은 벌써 pop-up menu 용의 갱신 핸들러를 작성했을 때에 사용하고 있는ID입니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl07-05/01.png  　다음에 CMainDlg 클래스에 가속기 테이블의 핸들을 멤버 변수로서 선언합니다. 이 변수는 CMainDlg 클래스의 constructor 으로 NULL로 설정합니다.   　OnInitDialog() 안에서는 AtlLoadAccelerators() 에 의해 가속기 테이블을 로드해 그 핸들을 멤버 변수에 대입합니다.   　마지막으로 PreTranslateMessage() 안에서 TranslateAccelerator() 를 호출합니다. |

▣ 다이얼로그 > 심플 다이얼로그

ATL은 CSimpleDialog 이라고 하는 다이얼로그용의 클래스를 준비해 있습니다. 이것은 주로 버젼 정보를 표시하는 것 같은 단순한 다이얼로그를 작성할 때에 사용합니다.

|  |
| --- |
| // 버젼 정보 다이얼로그 표시  CSimpleDialog<IDD\_ABOUTBOX> dlg;  dlg.DoModal(); |

CSimpleDialog 클래스를 사용하려면 제1템플릿 인수에 다이얼로그 자원의 것ID을 건네줍니다. 심플 다이얼로그는 DoModal()에 의해 모덜 다이얼로그로서 표시됩니다.   
  
CSimpleDialog 클래스에는 ID이IDOK, IDCANCEL, IDABORT, IDRETRY, IDIGNORE, IDYES, IDNO의 커멘드 핸들러가 준비되어 있습니다. 예를 들면 다이얼로그상에 몇 개의 ID를 가지는 버튼을 만들어 그 버튼을 누른면 다이얼로그는 닫습니다. 그 버튼의 ID는 DoModal()의 반환값 으로서 되돌아 옵니다.

▣ 위자드 > ATL/ WTL AppWizard

* 페이지 없음

▣ 위자드 > 스켈톤

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 지금까지 ATL/WTL AppWizard 를 사용하지 않고 Win32 Application 프로젝트로서 프로그램을 작성해 왔습니다만 , 여기서 ATL/WTL AppWizard 를 사용해 보겠습니다.   다음에 나타내는 것은 ATL/WTL AppWizard 으로 작성한 모달 다이얼로그의 스켈리턴(양식) 입니다. 작성하려면 ATL/WTL AppWizard 를 실행해 [Dialog Based] 를 선택해 [종료] 버튼을 누릅니다. 덧붙여 개행이나 코멘트 , 사용하지 않는 파라미터의 생략 등, 원 소스로부터 변경하고 있는 부분이 있습니다.   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h> |  |  | | --- | | // stdafx.cpp  #include "stdafx.h"  #if (\_ATL\_VER < 0x0700)  #include <atlimpl.cpp>  #endif //(\_ATL\_VER < 0x0700) |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // 메세지 맵  BEGIN\_MSG\_MAP(CMainDlg)  MESSAGE\_HANDLER(WM\_INITDIALOG, OnInitDialog)  COMMAND\_ID\_HANDLER(ID\_APP\_ABOUT, OnAppAbout)  COMMAND\_ID\_HANDLER(IDOK, OnOK)  COMMAND\_ID\_HANDLER(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(UINT, WPARAM, LPARAM, BOOL&)  {  // 스크린의 중앙에 배치  CenterWindow();  // 아이콘 설정  HICON hIcon = (HICON)::LoadImage(  \_Module.GetResourceInstance(),  MAKEINTRESOURCE(IDR\_MAINFRAME),  IMAGE\_ICON,  ::GetSystemMetrics(SM\_CXICON),  ::GetSystemMetrics(SM\_CYICON),  LR\_DEFAULTCOLOR);  SetIcon(hIcon, TRUE);  HICON hIconSmall = (HICON)::LoadImage(  \_Module.GetResourceInstance(),  MAKEINTRESOURCE(IDR\_MAINFRAME),  IMAGE\_ICON,  ::GetSystemMetrics(SM\_CXSMICON),  ::GetSystemMetrics(SM\_CYSMICON),  LR\_DEFAULTCOLOR);  SetIcon(hIconSmall, FALSE);  return TRUE;  }  LRESULT OnAppAbout(WORD, WORD, HWND, BOOL&)  {  // 버젼 정보 다이얼로그 표시  CSimpleDialog<IDD\_ABOUTBOX, FALSE> dlg;  dlg.DoModal();  return 0;  }  LRESULT OnOK(WORD, WORD wID, HWND, BOOL&)  {  EndDialog(wID);  return 0;  }  LRESULT OnCancel(WORD, WORD wID, HWND, BOOL&)  {  EndDialog(wID);  return 0;  }  }; |  |  | | --- | | // ModalDialog.cpp  #include "stdafx.h"  #include <atlframe.h>  #include <atlctrls.h>  #include <atldlgs.h>  #include <atlctrlw.h>  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(  HINSTANCE hInstance,  HINSTANCE /\*hPrevInstance\*/,  LPTSTR lpstrCmdLine,  int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  // NT4.0이후에 실행하는 경우는 , 대신에 다음의 호출을 사용하는 것으로 프리 thread화  // 시킬 수가 있다. 이것은 임의의RPCthread가 불려 가는 것을 의미한다.  // HRESULT hRes = ::CoInitializeEx(NULL, COINIT\_MULTITHREADED);  ATLASSERT(SUCCEEDED(hRes));  // 이것은Microsoft Layer for Unicode (MSLU) 이 사용되었을 때의  // ATL윈도우 thunking 문제를 해결하는  ::DefWindowProc(NULL, 0, 0, 0L);  // 다른 컨트롤을 서포트하기 위한 플래그를 추가  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_BAR\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　이 소스로 새로운 부분이라고 말하면 stdafx.cpp 파일 내와 \_tWinMain() 안입니다.   　stdafx.cpp파일에서는 ATL7.0 보다 낡은 버젼을 사용하고 있는 경우에 atlimpl.cpp 라고 하는 파일을 인클루드 하도록 하고 있습니다. atlimpl.cpp 에서는 \_ATL\_MIN\_CRT이 정의되고 있는 경우에 독자적인 WinMainCRTStartup() 등을 호출하도록 하고 있습니다. 이것에 의해 ATL는 가능한 한 C 런타임 라이브러리(CRT) 를 사용하지 않게 되어 프로그램의 파일 사이즈가 작아집니다.   덧붙여ATL/WTL AppWizard 로 작성한 프로젝트의 릴리스 빌드에서는 디폴트로\_ATL\_MIN\_CRT가 정의되고 있습니다.   \_tWinMain()에서는 COM 초기화 함수나 MSLU를 위한 코드가 추가되고 있습니다.   　그런데 ATL/WTL AppWizard 그리고 작성한 프로젝트의 자원은 디폴트로 영문판입니다. 이것을 일본어로 하기 위해서는 resource file(확장자(extension).rc) 를 텍스트 모드로 열어 이하의 부분：   |  | | --- | | /////////////////////////////////////////////////////////////////////////////  // English (U.S.) resources  #if !defined(AFX\_RESOURCE\_DLL) || defined(AFX\_TARG\_ENU)  #ifdef \_WIN32  LANGUAGE LANG\_ENGLISH, SUBLANG\_ENGLISH\_US  #pragma code\_page(1252)  #endif //\_WIN32 |   을 다음과 같이 고쳐 씁니다.   |  | | --- | | /////////////////////////////////////////////////////////////////////////////  // 일본어 resources  #if !defined(AFX\_RESOURCE\_DLL) || defined(AFX\_TARG\_JPN)  #ifdef \_WIN32  LANGUAGE LANG\_JAPANESE, SUBLANG\_DEFAULT  #pragma code\_page(932)  #endif //\_WIN32 | |

▣ 위자드 > \_ATL\_MIN\_CRT에 대해서

* 페이지없음

▣ 표준컨트룰러 > 버튼

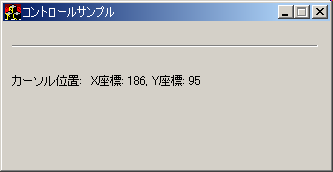
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WTL에서 표준 컨트롤의 버튼 컨트롤(푸쉬 버튼 , 체크 박스 , 라디오 버튼)을 CButtonT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는CButtonT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlctrls.h  typedef CButtonT<ATL::CWindow> CButton; |   이것은 CButton클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   　WTL의CButton클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CButton클래스를 사용하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl09-01/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해(때문에) |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CButton m\_radio\_addicon; // [아이콘 있어]라디오 버튼  CButton m\_radio\_removeicon; // [아이콘 없음]라디오 버튼  CButton m\_check\_topmost; // [항상 앞에 표시]체크 박스  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_SHOW, OnButtonShow)  COMMAND\_ID\_HANDLER\_EX(IDC\_CHECK\_TOPMOST, OnCheckTopmost)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_radio\_addicon = GetDlgItem(IDC\_RADIO\_ADDICON);  m\_radio\_removeicon = GetDlgItem(IDC\_RADIO\_REMOVEICON);  m\_check\_topmost = GetDlgItem(IDC\_CHECK\_TOPMOST);  m\_radio\_addicon.SetCheck(1);  return TRUE;  }  void OnButtonShow(UINT uNotifyCode, int nID, HWND hWndCtl){  UINT uIcon = MB\_OK;  if(m\_radio\_addicon.GetCheck())  uIcon |= MB\_ICONINFORMATION;  MessageBox(\_T("Hello, ATL/WTL"), \_T("메세지"), uIcon);  }  void OnCheckTopmost(UINT uNotifyCode, int nID, HWND hWndCtl){  SetWindowPos(m\_check\_topmost.GetCheck() ? HWND\_TOPMOST : HWND\_NOTOPMOST,  0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE | SWP\_SHOWWINDOW);  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 자원을 작성합니다. 다이얼로그에 버튼 컨트롤을 배치해 자원 ID를 다음과 같이 지정합니다.   |  |  | | --- | --- | | 컨트롤 명 | 자원 ID | | 푸쉬 버튼 | IDC\_BUTTON\_SHOW | | 라디오 버튼([아이콘 있어]용) | IDC\_RADIO\_ADDICON | | 라디오 버튼([아이콘 없음]용) | IDC\_RADIO\_REMOVEICON | | 체크 박스 | IDC\_CHECK\_TOPMOST |   　다음에 stdafx.h안에서는 CButton 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다. atlctrls.h 헤더에는 표준 컨트롤과 공용 컨트롤용의 클래스가 정의되고 있습니다.   　CMainDlg클래스에서는 우선 두 개의 라디오 버튼과 하나의 체크 박스용으로 CButton 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것들을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   　다음에 자원 ID가 IDC\_BUTTON\_SHOW와 IDC\_CHECK\_TOPMOST의 커멘드 메세지 핸들러로서 각각 OnButtonShow() , OnCheckTopmost()를 추가합니다.   OnButtonShow()는 푸쉬 버튼을 클릭했을 때에 불려 갑니다. 이 핸들러 함수에서는 라디오 버튼 상태를 CButtonT::GetCheck()에 의해 취득해 [아이콘 있어]가 선택되고 있었을 경우는 MB\_ICONINFORMATION을 지정해 메시지 박스를 표시합니다.   OnCheckTopmost()는 체크 박스를 클릭했을 때에 불려 갑니다. 이 핸들러 함수에서는 체크 박스 상태를 CButtonT::GetCheck()에 의해 취득해 SetWindowPos()에 의해 윈도우의Z오더를 변경합니다.   　덧붙여 CButtonT 클래스는 Click()이라고 하는 멤버 함수를 준비해 있습니다. 이것은 클릭을 시뮬레이트 합니다. |

▣ 표준컨트룰러 > 스태틱

　WTL에서는 표준 컨트롤의 스태틱 컨트롤을 CStaticT 이라고 하는 템플릿 클래스에서 캡슐화 하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CStaticT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CStaticT<ATL::CWindow> CStatic; |

이것은 CStatic 클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CStatic 클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CStatic 클래스를 사용해 경계선과 윈도우상의 커서 위치의 좌표를 표시하는 예입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CStatic m\_static\_border; // 경계선용  CStatic m\_static\_pos; // 커서 좌표 표시용  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_MOUSEMOVE(OnMouseMove)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_static\_border = GetDlgItem(IDC\_STATIC\_BORDER);  m\_static\_pos = GetDlgItem(IDC\_STATIC\_POS);  CRect rcBorder;  m\_static\_border.GetWindowRect(rcBorder);  m\_static\_border.SetWindowPos(NULL, 0, 0, rcBorder.Width(), 2,  SWP\_NOZORDER | SWP\_NOMOVE | SWP\_SHOWWINDOW);  return TRUE;  }  void OnMouseMove(UINT uFlags, CPoint ptClient){  CString strPos;  strPos.Format(\_T("X좌표: %d, Y좌표: %d"), ptClient.x, ptClient.y);  m\_static\_pos.SetWindowText(strPos);  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 자원을 작성합니다. 다이얼로그에 스태틱 컨트롤을 배치해 자원 ID를 다음과 같이 지정합니다. 덧붙여 경계선용 스태틱 컨트롤의 [스타일]에서는 디폴트에 가세해 [움푹한 곳]에 체크를 넣습니다.

|  |  |
| --- | --- |
| 컨트롤명 | 자원ID |
| 스태틱(경계선용) | IDC\_STATIC\_BORDER |
| 스태틱([커서 위치:]라벨용) | IDC\_STATIC(디폴트인 채 ) |
| 스태틱(커서 좌표 표시용) | IDC\_STATIC\_POS |

　다음에 stdafx.h 안에서는 CStatic 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   
  
　CMainDlg클래스에서는 우선 경계선과 커서 좌표 표시용으로 CStatic 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것들을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　다음에 경계선을 작성하기 위해서 WM\_INITDIALOG 메세지 핸들러로 SetWindowPos()를 사용해 경계선용 스태틱 컨트롤의 높이를 2 로 합니다.   
  
　다음에 WM\_MOUSEMOVE 메세지 핸들러를 추가해 클라이언트 영역상의 커서 위치를 스태틱 컨트롤로 설정해 있습니다.

▣ 표준컨트룰러 > 리스트 박스

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WTL에서는 표준 컨트롤의 리스트 박스 컨트롤을 CListBoxT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CListBoxT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlctrls.h  typedef CListBoxT<ATL::CWindow> CListBox; |   이것은 CListBox 클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   　WTL의 CListBox클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CListBox 클래스를 사용해 리스트 박스에 사용 가능한 TrueType 폰트명을 표시하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl09-03/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CListBox m\_list\_font; // 폰트명 표시용  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_HANDLER\_EX(IDC\_LIST\_FONT, LBN\_DBLCLK, OnListDblClick)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_list\_font = GetDlgItem(IDC\_LIST\_FONT);  // 사용 가능한 폰트명 열거  CClientDC dc(m\_hWnd);  EnumFontFamilies(dc, NULL,  (FONTENUMPROC)FontProc, (LPARAM)m\_list\_font.m\_hWnd);  return TRUE;  }  static int CALLBACK FontProc(ENUMLOGFONT \*lpelf,  NEWTEXTMETRIC \*lpntm, int nFontType, LPARAM lParam)  {  CListBox list = (HWND)lParam;  // TrueType폰트명만 리스트에 추가  if(nFontType & TRUETYPE\_FONTTYPE)  list.AddString(lpelf->elfLogFont.lfFaceName);  return 1;  }  void OnListDblClick(UINT uNotifyCode, int nID, HWND hWndCtl){  // 현재 선택되고 있는 아이템을 취득해 표시  int nIndex = m\_list\_font.GetCurSel();  if(nIndex != LB\_ERR){  CString strText;  m\_list\_font.GetText(nIndex, strText);  MessageBox(strText);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 자원을 작성합니다. 다이얼로그에 리스트 박스 컨트롤을 배치해 자원 ID를 다음과 같이 지정합니다.   |  |  | | --- | --- | | 컨트롤명 | 자원 ID | | 리스트 박스 | IDC\_LIST\_FONT |   　다음에 stdafx.h 안에서는 CListBox 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   　CMainDlg클래스에서는 우선 리스트 박스 컨트롤용으로 CListBox 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   　다음에 WM\_INITDIALOG 메세지 핸들러로 EnumFontFamilies()를 사용해 사용 가능한 폰트를 열거합니다. EnumFontFamilies()의 제3인수에는 콜백 함수인 FontProc()의 주소를 건네줍니다. 콜백 함수는 static 이기 때문에 FontProc()안에서는 CMainDlg 클래스의 멤버 변수등을 직접 사용할 수가 없습니다. 그런데 EnumFontFamilies()의 제4인수로 건네준 리스트 박스 컨트롤의 핸들을 사용해 재차 FontProc() 내에서 리스트 박스 컨트롤 오브젝트를 작성합니다.   　다음에 통지 코드가 LBN\_DBLCLK의 리스트 박스 컨트롤용 커멘드 메세지 핸들러를 추가합니다. 그 핸들러에서는 현재 선택되고 있는 아이템(폰트명) 을 취득해 메시지 박스에 표시합니다. |

▣ 표준컨트룰러 > 콤보박스

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WTL에서는 표준 컨트롤의 combobox 컨트롤을 CComboBoxT 라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CComboBoxT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlctrls.h  typedef CComboBoxT<ATL::CWindow> CComboBox; |   이것은 CComboBox 클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   　WTL의 CComboBox 클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CComboBox 클래스를 사용해 현재 디렉토리에 있는 파일 일람을 combobox 컨트롤에 표시하는 예입니다. combobox 컨트롤로 파일명을 선택하면 스태틱 컨트롤에 파일 사이즈를 표시합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl09-04/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CComboBox m\_combo\_dir; // 파일명 표시용  CStatic m\_static\_size; // 파일 사이즈 표시용  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_HANDLER\_EX(IDC\_COMBO\_DIR, CBN\_SELCHANGE, OnComboSelChange)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_combo\_dir = GetDlgItem(IDC\_COMBO\_DIR);  m\_static\_size = GetDlgItem(IDC\_STATIC\_SIZE);  m\_combo\_dir.Dir(DDL\_READWRITE, \_T("\*.\*"));  return TRUE;  }  void OnComboSelChange(UINT uNotifyCode, int nID, HWND hWndCtl){  int nIndex = m\_combo\_dir.GetCurSel();  if(nIndex != CB\_ERR){  CString strText;  m\_combo\_dir.GetLBText(nIndex, strText);  TCHAR szPath[\_MAX\_PATH];  LPTSTR lpFilePart;  GetFullPathName(strText,  sizeof(szPath)/sizeof(TCHAR), szPath, &lpFilePart);  HANDLE hFile = CreateFile(szPath, 0, 0, NULL,  OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  if(hFile != INVALID\_HANDLE\_VALUE){  DWORD dwSize = GetFileSize(hFile, NULL);  CloseHandle(hFile);  CString strSize;  strSize.Format(\_T("파일 사이즈: %ld아르바이트"), dwSize);  m\_static\_size.SetWindowText(strSize);  }  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 자원을 작성합니다. 다이얼로그에 combobox 컨트롤과 스태틱 컨트롤을 배치해 자원 ID를 다음과 같이 지정합니다. 덧붙여 combobox 컨트롤의 [스타일]에서는 디폴트에 가세해[타입]으로 [드롭 다운 리스트]를 선택해 [수평 오토 스크롤]에 체크를 넣습니다.   |  |  | | --- | --- | | 컨트롤명 | 자원 ID | | Combobox | IDC\_COMBO\_DIR | | 스태틱(사이즈 표시용) | IDC\_STATIC\_SIZE |   　다음에 stdafx.h안에서는 CComboBox 클래스나 CStatic 클래스를 사용하기 위해서 atlctrls.h 헤더를  인클루드 합니다.   　CMainDlg 클래스에서는 우선 combobox 컨트롤과 스태틱 컨트롤용으로 각각 CComboBox 클래스와 CStat ic클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   　다음에 WM\_INITDIALOG 메세지 핸들러로 Dir()를 사용해 combobox 컨트롤에 커런트 디렉토리내의 파일명 일람을 표시합니다.   　다음에 통지 코드가CBN\_SELCHANGE의 combobox 컨트롤용 커멘드 메세지 핸들러를 추가합니다. 그 핸들러에서는 GetLBText()에 의해 현재 선택되고 있는 아이템(파일명) 을 취득해 , 그 파일 사이즈를 스태틱 컨트롤에 표시합니다. |

▣ 표준컨트룰러 > 스크롤 바

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WTL에서는 표준 컨트롤의 스크롤 바 컨트롤을 CScrollBarT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CScrollBarT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlctrls.h  typedef CScrollBarT<ATL::CWindow> CScrollBar; |   이것은 CScrollBar 클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   　WTL의CScrollBar클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CScrollBar 클래스를 사용하는 예입니다. RGB각각의 스크롤 바를 조작해 색을 작성합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl09-05/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CScrollBar m\_scroll\_r; // [R]스크롤 바  CScrollBar m\_scroll\_g; // [G]스크롤 바  CScrollBar m\_scroll\_b; // [B]스크롤 바  CRect m\_rect; // 색 표시용 구형  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_HSCROLL(OnHScroll)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_scroll\_r = GetDlgItem(IDC\_SCROLLBAR\_R);  m\_scroll\_g = GetDlgItem(IDC\_SCROLLBAR\_G);  m\_scroll\_b = GetDlgItem(IDC\_SCROLLBAR\_B);  // 스크롤 바의 범위 설정  m\_scroll\_r.SetScrollRange(0, 255);  m\_scroll\_g.SetScrollRange(0, 255);  m\_scroll\_b.SetScrollRange(0, 255);  // 스크롤 바의 초기 위치 설정  m\_scroll\_r.SetScrollPos(0);  m\_scroll\_g.SetScrollPos(0);  m\_scroll\_b.SetScrollPos(0);  // 색 표시용 구형을 취득  CStatic view = GetDlgItem(IDC\_STATIC\_VIEW);  view.GetWindowRect(m\_rect);  ScreenToClient(m\_rect);  return TRUE;  }  void OnHScroll(int nSBCode, short nPos, HWND hWnd){  // 조작된 스크롤 바의 현재 위치를 취득  CScrollBar scroll = hWnd;  int color = scroll.GetScrollPos();  switch(nSBCode){  case SB\_PAGELEFT:  color -= 15;  case SB\_LINELEFT:  color = max(0, color - 1);  break;  case SB\_PAGERIGHT:  color += 15;  case SB\_LINERIGHT:  color = min(255, color + 1);  break;  case SB\_LEFT:  color = 0;  break;  case SB\_RIGHT:  color = 255;  break;  case SB\_THUMBPOSITION:  case SB\_THUMBTRACK:  color = nPos;  break;  }  // 새로운 위치로 설정  scroll.SetScrollPos(color);  // 색 표시용 구형을 갱신  InvalidateRect(m\_rect);  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CBrush brush;  brush.CreateSolidBrush(RGB(m\_scroll\_r.GetScrollPos(),  m\_scroll\_g.GetScrollPos(), m\_scroll\_b.GetScrollPos()));  dc.FillRect(m\_rect, brush);  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 자원을 작성합니다. 다이얼로그에 스크롤 바 컨트롤과 스태틱 컨트롤을 배치해 자원 ID를 다음과 같이 지정합니다. 덧붙여 색 표시용의 스태틱 컨트롤의 프롭퍼티에서는[가시]의 체크를 뗍니다.   |  |  | | --- | --- | | 컨트롤명 | 자원ID | | 스크롤 바([R]용) | IDC\_SCROLLBAR\_R | | 스크롤 바([G]용) | IDC\_SCROLLBAR\_G | | 스크롤 바([B]용) | IDC\_SCROLLBAR\_B | | 스태틱([R]라벨용) | IDC\_STATIC(디폴트인 채 ) | | 스태틱([G]라벨용) | IDC\_STATIC(디폴트인 채 ) | | 스태틱([B]라벨용) | IDC\_STATIC(디폴트인 채 ) | | 스태틱(색 표시용) | IDC\_STATIC\_VIEW |   　다음에 stdafx.h 안에서는 CScrollBar클래스를 사용하기 위해서 atlctrls.h헤더를 인클루드 합니다.   　CMainDlg클래스에서는 우선 스크롤 바 컨트롤용으로 CScrollBar 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다. 그리고 색을 표시하는 구형을 나타내는 CRect 클래스의 인스턴스도 멤버 변수로서 선언해 둡니다.   　다음에 WM\_INITDIALOG 메세지 핸들러로 스크롤 바 컨트롤의 범위와 초기 위치를 설정해 색 표시용의 스태틱 컨트롤 구형을 취득합니다.   　다음에 스크롤 바의 WM\_HSCROLL 메세지 핸들러를 추가합니다. 이 핸들러에서는 스크롤 바 컨트롤의 위치를 설정해 색 표시용 구형을 갱신합니다.   　마지막으로 WM\_PAINT 메세지 핸들러를 추가해 색을 묘화 합니다. |

▣ 표준컨트룰러 > 에디터

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WTL에서는 표준 컨트롤의 에디트 컨트롤을 CEditT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CEditT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlctrls.h  typedef CEditT<ATL::CWindow> CEdit; |   이것은 CEdit 클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   　WTL의 CEdit 클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CEdit 클래스를 사용하는 예입니다. 에디트 컨트롤에 입력한 문자를 다이얼로그에 묘화 합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl09-06/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  enum { MAX\_DRAWTEXT = 256};  CEdit m\_edit\_text; // 문자 설정용 에디트  CRect m\_rect; // 문자 표시용 구형  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_HANDLER\_EX(IDC\_EDIT\_TEXT, EN\_CHANGE, OnEditChange)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_edit\_text = GetDlgItem(IDC\_EDIT\_TEXT);  m\_edit\_text.SetLimitText(MAX\_DRAWTEXT - 1);  // 문자 표시용 구형을 취득  CStatic view = GetDlgItem(IDC\_STATIC\_VIEW);  view.GetWindowRect(m\_rect);  ScreenToClient(m\_rect);  return TRUE;  }  void OnEditChange(UINT uNotifyCode, int nID, HWND hWndCtl){  InvalidateRect(m\_rect, FALSE);  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  // 흰 구형을 묘화  CBrush brush;  brush.CreateSolidBrush(RGB(255, 255, 255));  dc.FillRect(m\_rect, brush);  // 텍스트색을 파랑으로 설정  dc.SetTextColor(RGB(0, 0, 255));  HFONT hOldFont = dc.SelectFont(AtlGetDefaultGuiFont());  // 클라이언트 구형을 취득해 , 캐릭터 라인을 묘화  TCHAR szText[MAX\_DRAWTEXT];  m\_edit\_text.GetWindowText(szText, MAX\_DRAWTEXT);  dc.DrawText(szText, lstrlen(szText),  m\_rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 자원을 작성합니다. 다이얼로그에 에디트 컨트롤과 스태틱 컨트롤을 배치해 자원 ID를 다음과 같이 지정합니다. 덧붙여 문자 표시용의 스태틱 컨트롤의 프롭퍼티에서는[가시]의 체크를 뗍니다.   |  |  | | --- | --- | | 컨트롤명 | 자원 ID | | 에디트 | IDC\_EDIT\_TEXT | | 스태틱(문자 표시용) | IDC\_STATIC\_VIEW |   　다음에 stdafx.h안에서는 CEdit 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   　CMainDlg클래스에서는 우선 enum에 의해 최대 문자수를 나타내는 MAX\_DRAWTEXT을 정의합니다. 다음에 에디트용으로 CEdit 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다. 그리고 문자를 표시하는 구형을 나타내는 CRect 클래스의 인스턴스도 멤버 변수로서 선언해 둡니다.   　다음에 WM\_INITDIALOG 메세지 핸들러로 에디트 컨트롤의 최대 문자수를 설정해 문자 표시용의 스태틱 컨트롤 구형을 취득합니다.   　다음에 통지 코드가 EN\_CHANGE의 에디트 컨트롤용 커멘드 메세지 핸들러를 추가합니다. 그 핸들러에서는 표시 구형을 갱신합니다.   　마지막으로 WM\_PAINT 메세지 핸들러를 추가해 문자를 묘화 합니다. |

▣ 공용 컨트룰러 > 툴팁

|  |  |
| --- | --- |
| |  | | --- | | 툴 팁 | |
| WTL에서는 공용 컨트롤의 툴 팁 컨트롤을CToolTipCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만,, 그 클래스는CToolTipCtrlT클래스의 base class로서 사용됩니다. atlctrls.h헤더에서는typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlctrls.h안  typedef CToolTipCtrlT<ATL::CWindow> CToolTipCtrl; |   이것은 CToolTipCtrl클래스는CWindow클래스의 상속 클래스인 것을 의미합니다.   　WTL의CToolTipCtrl클래스는 MFC의 동명의 클래스와 동등의 기능을 갖추고 있는 것 외에 확장 기능도 갖추고 있습니다. 이하에 나타내는 것은 CToolTipCtrl클래스를 사용하는 예입니다. 표시하는 툴 팁을 정적으로 설정하는 방법과 동적으로 설정하는 방법을 나타냅니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 정적인 툴 팁 |
| 우선 , 툴 팁을 정적으로 설정하는 방법입니다. 이하의 예는 다이얼로그상의 에디트 컨트롤에 마우스 커서를 실으면 「edit control」이라고 하는 툴 팁을 표시합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl10-01/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CEdit m\_edit\_tooltip; // 툴 팁 표시용  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_edit\_tooltip = GetDlgItem(IDC\_EDIT\_TOOLTIP);  // 툴 팁 컨트롤 작성  CToolTipCtrl tooltip;  tooltip.Create(m\_hWnd);  tooltip.Activate(TRUE);  // 에디트 컨트롤의 툴 팁 정보를 툴 팁 컨트롤에 추가  CToolInfo tiEdit(TTF\_SUBCLASS, m\_edit\_tooltip, 0, NULL, \_T("edit control"));  tooltip.AddTool(tiEdit);  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp내  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선, 자원을 작성합니다. 다이얼로그에 에디트 컨트롤을 배치해 자원ID를 다음과 같이 지정합니다. 덧붙여 에디트 컨트롤의[스타일]에서는 디폴트에 [수평 스크롤 바], [수직 스크롤 바],[수직 오토 스크롤],[개행을 허가]에 체크를 합니다.   |  |  | | --- | --- | | 컨트롤명 | 자원ID | | 에디트 | IDC\_EDIT\_TOOLTIP |   　다음에 stdafx.h안에서는 CToolTipCtrl클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   　다음에 CMainDlg클래스의 WM\_INITDIALOG 메세지 핸들러로 에디트 컨트롤을 초기화하고 툴 팁 컨트롤을 작성합니다. 그 후 에디트 컨트롤에 대한 툴 팁 정보를 작성해 그것을 툴 팁 컨트롤에 추가합니다.   이것으로 에디트 컨트롤에 마우스 커서를 갖다 대면 툴 팁이 표시됩니다.   　덧붙여 표시하는 툴 팁 캐릭터 라인을 복수 행으로 할 수도 있습니다. 그러기 위해서는 CToolTipCtrl클래스의 멤버 함수인 SetMaxTipWidth()을 사용해 툴 팁의 폭을 설정할 필요가 있습니다. 이렇게 하는 것으로 폭보다 긴 캐릭터 라인은 자동적으로 복수 행으로 표시됩니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl10-01/02.png   |  | | --- | | // OnInitDialog()  CToolTipCtrl tooltip;  tooltip.Create(m\_hWnd);  tooltip.Activate(TRUE);  // 툴 팁폭을 설정  tooltip.SetMaxTipWidth(80);  CToolInfo tiEdit(TTF\_SUBCLASS, m\_edit\_tooltip, 0, NULL, \_T("복수행의 툴 팁"));  tooltip.AddTool(tiEdit); |   그리고 캐릭터 라인의 임의의 장소에서 개행 하고 싶은 경우는 「\r\n」 으로 개행 할 수가 있습니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl10-01/03.png   |  | | --- | | // OnInitDialog()  CToolTipCtrl tooltip;  tooltip.Create(m\_hWnd);  tooltip.Activate(TRUE);  // 툴 팁폭을 설정  tooltip.SetMaxTipWidth(80);  CToolInfo tiEdit(TTF\_SUBCLASS, m\_edit\_tooltip, 0, NULL, \_T("복수행의\r\n툴 팁"));  tooltip.AddTool(tiEdit); | | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 동적인 툴 팁 |
| 다음에 , 툴 팁을 동적으로 설정하는 방법을 나타냅니다. 이하에 나타내는 것은 다이얼로그상의 에디트 컨트롤에 마우스 커서를 갖다 대면 에디트 컨트롤의 현재의 행수를 툴 팁으로 표시하는 예 입니다. 덧붙여 앞의 정적인 툴 팁을 작성한 프로그램의 CMainDlg 클래스만을 변경합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl10-01/04.png   |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CEdit m\_edit\_tooltip; // 툴 팁 표시용 에디트  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_CODE\_HANDLER\_EX(TTN\_GETDISPINFO, OnTooltipGetDispInfo)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_edit\_tooltip = GetDlgItem(IDC\_EDIT\_TOOLTIP);  // 툴 팁 컨트롤 작성  CToolTipCtrl tooltip;  tooltip.Create(m\_hWnd);  tooltip.Activate(TRUE);  // 에디트 컨트롤의 툴 팁 정보를 툴 팁 컨트롤에 추가  CToolInfo tiEdit(TTF\_SUBCLASS, m\_edit\_tooltip);  tooltip.AddTool(tiEdit);  return TRUE;  }  LRESULT OnTooltipGetDispInfo(LPNMHDR pnmh){  LPNMTTDISPINFO pntdi = (LPNMTTDISPINFO)pnmh;  if(pntdi->uFlags & TTF\_IDISHWND){  int ID = ::GetDlgCtrlID((HWND)pntdi->hdr.idFrom);  if(ID == IDC\_EDIT\_TOOLTIP){  CString strText;  strText.Format(\_T("%d행"), m\_edit\_tooltip.GetLineCount());  lstrcpy(pntdi->szText, strText);  }  }  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |   　우선 WM\_INITDIALOG 메세지 핸들러로 툴 팁 정보를 작성할 때 제3인수 이후를 생략 합니다. 이렇게 하면 제5인수에는 디폴트로LPSTR\_TEXTCALLBACK가 지정되어 툴 팁을 표시하는 타이밍에 통지 코드가TTN\_GETDISPINFO의 WM\_NOTIFY메세지가 발생하게 됩니다.   　다음에 통지 코드가TTN\_GETDISPINFO의 툴 팁 컨트롤용 통지 메세지 핸들러를 추가합니다. 그 핸들러에서는 에디트 컨트롤의 현재의 행수를 취득해 툴 팁으로 해서 표시하고 있습니다. | |

▣ 공용 컨트룰러 > 풍선 툴팁

**풍선 툴팁**

Internet Explorer 5 이상이 인스톨 되어 있는 시스템에서는 풍선 툴 팁을 사용할 수 있습니다. 풍선 툴 팁이란 아래 그림과 같은 형태를 한 툴 팁입니다. WTL의CToolTipCtrl클래스는 풍선 툴 팁을 위한 멤버 함수를 준비해 있습니다.   
  
이하에 나타내는 것은 CToolTipCtrl클래스를 사용해 풍선 툴 팁을 표시하는 예입니다. 다이얼로그상의 버튼 컨트롤에 마우스 커서를 갖다대면 「풍선 툴 팁 샘플」이라고 하는 툴 팁을 표시합니다.



|  |
| --- |
| // stdafx.h  #define \_WIN32\_IE 0x0500  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 툴 팁 컨트롤 작성  CToolTipCtrl tooltip;  tooltip.Create(m\_hWnd, NULL, NULL, TTS\_BALLOON);  tooltip.Activate(TRUE);  tooltip.SetTitle(1, \_T("풍선 툴 팁"));  // 버튼 컨트롤의 툴 팁 정보를 툴 팁 컨트롤에 추가  CToolInfo tiBtn(TTF\_SUBCLASS,  GetDlgItem(IDOK), 0, NULL, \_T("풍선 툴 팁의 샘플"));  tooltip.AddTool(tiBtn);  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 자원을 작성합니다. 다이얼로그에 버튼 컨트롤을 배치해 자원ID을 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 자원ID |
| 푸쉬 버튼 | IDOK |

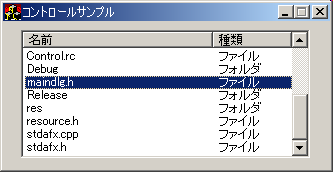
　다음에 stdafx.h 에서\_WIN32\_IE를 0x0500에 정의합니다. 이것에 의해 CToolTipCtrl::SetTitle()가 사용 가능하게 됩니다. 게다가 CToolTipCtrl 클래스를 사용하기 위해서 atlctrls.h헤더를 포함 합니다.   
  
　CMainDlg클래스의 WM\_INITDIALOG 메세지 핸들러 에서는 툴 팁 컨트롤을 작성할 경우에 Create()의 제4인수에 TTS\_BALLOON을 지정해 SetTitle() 그리고 풍선 툴 팁에 아이콘과 타이틀을 설정해 있습니다.

▣ 공용 컨트룰러 > 리스트뷰

**< 리스트 뷰 >**  
  
WTL에서 공용 컨트롤의 리스트 뷰 컨트롤을 CListViewCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는CListViewCtrlT클래스의 base class으로서 사용됩니다. atlctrls.h헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h안  typedef CListViewCtrlT<ATL::CWindow> CListViewCtrl; |

이것은 CListViewCtrl클래스는 CWindow클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CListViewCtrl클래스는 MFC의 CListCtrl클래스와 동등의 기능을 갖추고 있는 것 외에 확장 기능도 갖추고 있습니다. 이하에 나타내는 것은 CListViewCtrl클래스를 사용하는 예입니다. 현재 디렉토리의 폴더명과 파일명을 리스트 뷰에 추가해 아이템을 더블 클릭 하면 이름과 종류를 메시지 박스로 표시합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CListViewCtrl m\_list\_file;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_HANDLER\_EX(IDC\_LIST\_FILE, NM\_DBLCLK, OnListDblClick)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_list\_file = GetDlgItem(IDC\_LIST\_FILE);  m\_list\_file.SetExtendedListViewStyle(LVS\_EX\_INFOTIP | LVS\_EX\_FULLROWSELECT);  // 리스트 뷰 컨트롤에 컬럼 삽입  CRect rcList;  m\_list\_file.GetWindowRect(rcList);  int nScrollWidth = GetSystemMetrics(SM\_CXVSCROLL);  int n3DEdge = GetSystemMetrics(SM\_CXEDGE);  m\_list\_file.InsertColumn(0, \_T("이름"), LVCFMT\_LEFT, 190, -1);  m\_list\_file.InsertColumn(1, \_T("종류"), LVCFMT\_LEFT,  rcList.Width() - 190 - nScrollWidth - n3DEdge \* 2, -1);  // 리스트 뷰 컨트롤에 아이템 추가  CFindFile find;  if(find.FindFile()){  do{  if(!find.IsDots()){  int nIndex = m\_list\_file.GetItemCount();  m\_list\_file.AddItem(nIndex, 0, find.GetFileName());  m\_list\_file.AddItem(nIndex, 1,  find.IsDirectory() ? \_T("폴더") : \_T("파일"));  }  }while(find.FindNextFile());  }  return TRUE;  }  LRESULT OnListDblClick(LPNMHDR pnmh){  LPNMITEMACTIVATE pnmia = (LPNMITEMACTIVATE)pnmh;  CString strItem;  m\_list\_file.GetItemText(pnmia->iItem, 0, strItem);  CString strSubItem;  m\_list\_file.GetItemText(pnmia->iItem, 1, strSubItem);  MessageBox(strItem + \_T("은") + strSubItem + \_T("입니다. "), \_T("아이템"));  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

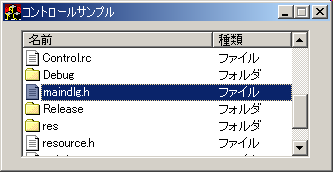
　우선 자원을 작성합니다. 다이얼로그에 리스트 뷰 컨트롤을 배치해 자원ID를 다음과 같이 지정합니다. 덧붙여 리스트 뷰 컨트롤의 [스타일]에서는 디폴트에 다가 [표시]로 [리포트]를 선택하고 [단일 선택]에 체크를 넣습니다.

|  |  |
| --- | --- |
| 컨트롤명 | 자원ID |
| 리스트뷰 | IDC\_LIST\_FILE |

　다음에 stdafx.h 에서는 CListViewCtrl 클래스를 사용하기 위해서 atlctrls.h헤더를 포함 합니다.   
  
　CMainDlg클래스에서는 우선 리스트 뷰 컨트롤 용으로 CListViewCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러에서 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러로 리스트 뷰 컨트롤의 확장 스타일을 설정해서 컬럼을 삽입하고 CFindFile 클래스를 사용해 현재 디렉토리의 폴더와 파일을 조사 합니다.   
  
　CFindFile 클래스는 atlmisc.h에 정의되고 있어 이름이 비슷한 MFC의 CFileFind 클래스와 거의 같은 멤버 함수를 준비해 있습니다.   
  
　다음으로 통지 코드가 NM\_DBLCLK의 리스트 뷰 컨트롤 용 통지 메세지 핸들러를 추가합니다. 그 핸들러 에서는 더블 클릭 된 아이템의 텍스트를 취득해서 메시지 박스로 표시하고 있습니다.

**▣ 공용 컨트룰러 > 이미지 리스트**

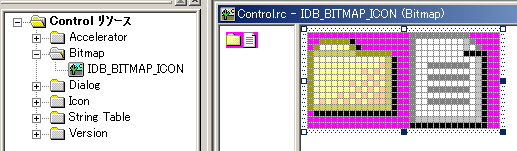
WTL에서는 이미지 리스트를 CImageList 이라고 하는 클래스에서 캡슐화하고 있습니다. 　WTL의 CImageList 클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 아래에 나타내는 것은 CImageList클래스를 사용하는 예입니다. 전회의 리스트 뷰 컨트롤에 아이콘을 추가합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CListViewCtrl m\_list\_file;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_HANDLER\_EX(IDC\_LIST\_FILE, NM\_DBLCLK, OnListDblClick)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_list\_file = GetDlgItem(IDC\_LIST\_FILE);  m\_list\_file.SetExtendedListViewStyle(LVS\_EX\_INFOTIP | LVS\_EX\_FULLROWSELECT);  // 이미지 리스트를 작성  CImageList il;  il.Create(16, 16, ILC\_COLOR8 | ILC\_MASK, 2, 1);  // 비트 맵 자원을 이미지 리스트에 추가  CBitmap bmp;  bmp.LoadBitmap(IDB\_BITMAP\_ICON);  il.Add(bmp, RGB(255, 0, 255));  // 이미지 리스트를 리스트 뷰 컨트롤로 설정  m\_list\_file.SetImageList(il, LVSIL\_SMALL);  // 리스트 뷰 컨트롤에 컬럼 삽입  CRect rcList;  m\_list\_file.GetWindowRect(rcList);  int nScrollWidth = GetSystemMetrics(SM\_CXVSCROLL);  int n3DEdge = GetSystemMetrics(SM\_CXEDGE);  m\_list\_file.InsertColumn(0, \_T("이름"), LVCFMT\_LEFT, 190, -1);  m\_list\_file.InsertColumn(1, \_T("종류"), LVCFMT\_LEFT,  rcList.Width() - 190 - nScrollWidth - n3DEdge \* 2, -1);  // 리스트 뷰 컨트롤에 아이템 추가  CFindFile find;  if(find.FindFile()){  do{  if(!find.IsDots()){  int nIndex = m\_list\_file.GetItemCount();  BOOL bDir = find.IsDirectory();  m\_list\_file.AddItem(nIndex, 0, find.GetFileName(), bDir ? 0 : 1);  m\_list\_file.AddItem(nIndex, 1,  bDir ? \_T("폴더") : \_T("파일"));  }  }while(find.FindNextFile());  }  return TRUE;  }  LRESULT OnListDblClick(LPNMHDR pnmh){  LPNMITEMACTIVATE pnmia = (LPNMITEMACTIVATE)pnmh;  CString strItem;  m\_list\_file.GetItemText(pnmia->iItem, 0, strItem);  CString strSubItem;  m\_list\_file.GetItemText(pnmia->iItem, 1, strSubItem);  MessageBox(strItem + \_T("은") + strSubItem + \_T("입니다. "), \_T("아이템"));  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

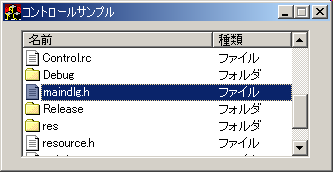
　우선 프로젝트에 이미지 리스트용의 비트맵 자원을 추가합니다. 자원 ID는 IDB\_BITMAP\_ICON 으로 하며 16x16사이즈의 화상을 2개 준비합니다. (즉 비트맵의 사이즈는 32x16 )  
  
  
  
　다음에 WM\_INITDIALOG 메시지핸들러로 CImageList 클래스의 인스턴스를 작성하여 비트맵 자원을 추가해 리스트 뷰로 설정합니다. 그리고 AddItem()에 의해 리스트 뷰에 아이템을 추가할 때에 제4 인수로 이미지 리스트의 인덱스를 지정합니다. 아이템이 폴더의 경우는 0, 파일의 경우는 1 입니다.

▣ 공용 컨트룰러 > 헤더

**< 헤더 >**　  
  
WTL에서 공용 컨트롤의 헤더 컨트롤을 CHeaderCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CHeaderCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CHeaderCtrlT<ATL::CWindow> CHeaderCtrl; |

이것은 CHeaderCtrl클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CHeaderCtrl클래스는 MFC의 동명의 클래스와 동등의 기능을 갖추고 있는 것 외에 확장 기능도 갖추고 있습니다. 이하에 나타내는 것은 CHeaderCtrl클래스를 사용하는 예입니다. 전회 이미지 리스트를 추가한 리스트 뷰 컨트롤의 헤더 부분을 오른쪽 클릭하면 헤더 아이템 명을 메시지 박스로 표시합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CListViewCtrl m\_list\_file;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_HANDLER\_EX(IDC\_LIST\_FILE, NM\_DBLCLK, OnListDblClick)  NOTIFY\_HANDLER\_EX(0, NM\_RCLICK, OnHeaderRClick)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_list\_file = GetDlgItem(IDC\_LIST\_FILE);  m\_list\_file.SetExtendedListViewStyle(LVS\_EX\_INFOTIP | LVS\_EX\_FULLROWSELECT);  // 이미지 리스트를 작성  CImageList il;  il.Create(16, 16, ILC\_COLOR8 | ILC\_MASK, 2, 1);  // 비트 맵 자원을 이미지 리스트에 추가  CBitmap bmp;  bmp.LoadBitmap(IDB\_BITMAP\_ICON);  il.Add(bmp, RGB(255, 0, 255));  // 이미지 리스트를 리스트뷰콘트로르로 설정  m\_list\_file.SetImageList(il, LVSIL\_SMALL);  // 리스트뷰콘트로르에 컬럼 삽입  CRect rcList;  m\_list\_file.GetWindowRect(rcList);  int nScrollWidth = GetSystemMetrics(SM\_CXVSCROLL);  int n3DEdge = GetSystemMetrics(SM\_CXEDGE);  m\_list\_file.InsertColumn(0, \_T("이름"), LVCFMT\_LEFT, 190, -1);  m\_list\_file.InsertColumn(1, \_T("종류"), LVCFMT\_LEFT,  rcList.Width() - 190 - nScrollWidth - n3DEdge \* 2, -1);  // 리스트뷰콘트로르에 아이템 추가  CFindFile find;  if(find.FindFile()){  do{  if(!find.IsDots()){  int nIndex = m\_list\_file.GetItemCount();  BOOL bDir = find.IsDirectory();  m\_list\_file.AddItem(nIndex, 0, find.GetFileName(), bDir ? 0 : 1);  m\_list\_file.AddItem(nIndex, 1,  bDir ? \_T("폴더") : \_T("파일"));  }  }while(find.FindNextFile());  }  return TRUE;  }  LRESULT OnListDblClick(LPNMHDR pnmh){  LPNMITEMACTIVATE pnmia = (LPNMITEMACTIVATE)pnmh;  CString strItem;  m\_list\_file.GetItemText(pnmia->iItem, 0, strItem);  CString strSubItem;  m\_list\_file.GetItemText(pnmia->iItem, 1, strSubItem);  MessageBox(strItem + \_T("은") + strSubItem + \_T("입니다. "), \_T("아이템"));  return 0;  }  LRESULT OnHeaderRClick(LPNMHDR pnmh){  CHeaderCtrl header = pnmh->hwndFrom;  CPoint pt(::GetMessagePos());  header.ScreenToClient(&pt);  HDHITTESTINFO hhi;  hhi.pt = pt;  if(header.HitTest(&hhi) != -1){  const int cchMax = 128;  TCHAR szText[cchMax];  HDITEM item = {HDI\_TEXT};  item.pszText = szText;  item.cchTextMax = cchMax;  header.GetItem(hhi.iItem, &item);  CString strMsg;  strMsg.Format(\_T("인덱스%d의 헤더 아이템명은%s입니다. "),  hhi.iItem, item.pszText);  MessageBox(strMsg, \_T("아이템"));  }  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 메세지 맵에 통지 코드가 NM\_RCLICK의 통지 메세지 엔트리를 추가합니다. 덧붙여 리포트 스타일의 리스트 뷰에 부속되는 헤더 컨트롤의 ID는 0 입니다.   
  
　다음에 통지 코드가 NM\_RCLICK의 통지 메세지 핸들러에서는 ::GetMessagePos()에 의해 오른쪽 클릭된 위치를 취득해 그 위치에 있는 헤더 아이템의 인덱스를 HitTest()로 취득합니다. 만약 그 위치에 아이템이 있으면 헤더 아이템명을 취득해 메시지 박스로 표시하고 있습니다.

▣ 공용 컨트룰러 > 트리 뷰

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **< 트리 뷰 >**  WTL에서 고용 컨트롤의 트리 뷰 컨트롤을 CTreeViewCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CTreeViewCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlctrls.h  typedef CTreeViewCtrlT<ATL::CWindow> CTreeViewCtrl; |   이것은 CTreeViewCtrl클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   　WTL의 CTreeViewCtrl클래스는 MFC의 CTreeCtrl클래스와 동등의 기능을 갖추고 있는 것 외에 확장 기능도 갖추고 있습니다. 이하에 나타내는 것은 CTreeViewCtrl클래스를 사용하는 예 입니다. 트리 뷰 컨트롤의 아이템을 클릭하면 그 아이템에 관한 정보를 static 컨트롤에 표시합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl10-06/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  enum {SHELL = 1, COMCTL, WINDIR, SYSDIR}; // 트리 아이템 식별자    CTreeViewCtrl m\_tree\_sysinfo;  CStatic m\_static\_display; // 정보 표시용  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_HANDLER\_EX(IDC\_TREE\_SYSINFO, NM\_CLICK, OnTreeClick)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_tree\_sysinfo = GetDlgItem(IDC\_TREE\_SYSINFO);  m\_static\_display = GetDlgItem(IDC\_STATIC\_DISPLAY);  HTREEITEM hSysInfo =  m\_tree\_sysinfo.InsertItem(\_T("시스템 정보"), TVI\_ROOT, TVI\_LAST);  HTREEITEM hDllVer =  m\_tree\_sysinfo.InsertItem(\_T("DLL버젼"), hSysInfo, TVI\_LAST);  HTREEITEM hItem;  hItem = m\_tree\_sysinfo.InsertItem(\_T("shell32.dll"), hDllVer, TVI\_LAST);  m\_tree\_sysinfo.SetItemData(hItem, SHELL);  hItem = m\_tree\_sysinfo.InsertItem(\_T("comctl32.dll"), hDllVer, TVI\_LAST);  m\_tree\_sysinfo.SetItemData(hItem, COMCTL);  HTREEITEM hSysDir =  m\_tree\_sysinfo.InsertItem(\_T("디렉토리"), hSysInfo, TVI\_LAST);  hItem = m\_tree\_sysinfo.InsertItem(\_T("Windows"), hSysDir, TVI\_LAST);  m\_tree\_sysinfo.SetItemData(hItem, WINDIR);  hItem = m\_tree\_sysinfo.InsertItem(\_T("시스템"), hSysDir, TVI\_LAST);  m\_tree\_sysinfo.SetItemData(hItem, SYSDIR);  return TRUE;  }  LRESULT OnTreeClick(LPNMHDR pnmh){  // 클릭된 아이템 핸들 취득  UINT uFlag;  CPoint pt = ::GetMessagePos();  m\_tree\_sysinfo.ScreenToClient(&pt);  HTREEITEM hItem = m\_tree\_sysinfo.HitTest(pt, &uFlag);  if(hItem == NULL || !(uFlag & TVHT\_ONITEM))  return 0;  // 아이템 데이터 취득  CString strInfo;  DWORD dwData = m\_tree\_sysinfo.GetItemData(hItem);  switch(dwData){  case SHELL:  {  // shell32.dll의 버젼 취득  DWORD dwMajor, dwMinor;  AtlGetShellVersion(&dwMajor, &dwMinor);  strInfo.Format(\_T("Version %d.%02d"), dwMajor, dwMinor);  break;  }  case COMCTL:  {  // comctl32.dll의 버젼 취득  DWORD dwMajor, dwMinor;  AtlGetCommCtrlVersion(&dwMajor, &dwMinor);  strInfo.Format(\_T("Version %d.%02d"), dwMajor, dwMinor);  break;  }  case WINDIR:  {  // Windows폴더 패스 취득  TCHAR szPath[MAX\_PATH];  GetWindowsDirectory(szPath, MAX\_PATH);  strInfo = szPath;  break;  }  case SYSDIR:  {  // 시스템 폴더 패스 취득  TCHAR szPath[MAX\_PATH];  GetSystemDirectory(szPath, MAX\_PATH);  strInfo = szPath;  break;  }  default:  break;  }  m\_static\_display.SetWindowText(strInfo);  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 자원을 작성합니다. 다이얼로그에 트리 뷰 컨트롤과 스태틱 컨트롤을 배치해 자원ID를 다음과 같이 지정합니다. 덧붙여 트리 뷰 컨트롤의 [스타일]에서는 디폴트에다가 [버튼 있음], [선을 표시], [최상정도에도 선을 표시]에 체크를 넣습니다.   |  |  | | --- | --- | | 컨트롤명 | 자원ID | | 트리뷰 | IDC\_TREE\_SYSINFO | | 스태틱 | IDC\_STATIC\_DISPLAY |   　다음에 stdafx.h안에서는 CTreeViewCtrl클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   　CMainDlg클래스에서는 우선 enum에 의해 트리 아이템을 식별하기 위한 수치를 정의합니다. 다음에 트리 뷰 컨트롤용으로 CTreeViewCtrl클래스 스태틱 컨트롤용으로 CStatic 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것들을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   　다음에 WM\_INITDIALOG 메세지 핸들러로 트리 뷰 컨트롤에 아이템을 삽입합니다. 여기서 방금 전 enum에 의해 정의한 식별자를SetItemData()로 아이템으로 설정합니다. 설정하지 않는 경우는 InsertItem() 그리고 아이템을 삽입했을 때에 이미 0 이 설정되어 있습니다. (이 때문에 식별자에는 0 이외를 사용하고 있습니다. )  　다음에 , 통지 코드가NM\_CLICK의 트리뷰콘트로르용 통지 메세지 핸들러를 추가합니다. 그 핸들러에서는 클릭된 아이템의 식별자를 취득해 , 그 값에 따라 정보를 스태틱 컨트롤에 표시하고 있습니다. |

**▣ 공용 컨트룰러 > 확장 트리 뷰**

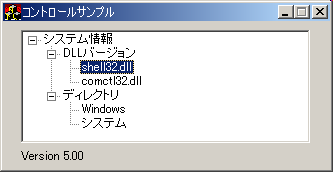
WTL에서 고용 컨트롤의 트리 뷰 컨트롤을 캡슐화하기 위해서 CTreeViewCtrlT 클래스를 준비해 있습니다만 CTreeViewCtrlExT 이라고 하는 클래스도 준비해 있습니다. 이 클래스는 CTreeViewCtrlT 클래스와 같은 템플릿 클래스입니다. CTreeViewCtrlExT 클래스는 CTreeViewCtrlT 클래스의 상속 클래스 이며 CTreeViewCtrlExT 클래스의 템플릿 인수는 base class인 CTreeViewCtrlT 클래스에 건네 받습니다. atlctrls.h헤더에서는 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  template <class TBase>  class CTreeViewCtrlExT : public CTreeViewCtrlT< TBase >  {  ...  ...  typedef CTreeViewCtrlExT<ATL::CWindow> CTreeViewCtrlEx; |

CTreeViewCtrlExT 클래스는 CTreeItemT 이라고 하는 템플릿 클래스를 사용합니다. CTreeItemT 클래스는 트리 아이템 핸들HTREEITEM을 캡슐화합니다. 그리고 CTreeItemT 클래스는 CTreeViewCtrlExT 클래스에의 포인터를 멤버 변수로서 가집니다. CTreeItemT 클래스의 템플릿 인수는 이 멤버 변수인 CTreeViewCtrlExT 클래스에 건네 받습니다. atlctrls.h 헤더에서는 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  template <class TBase>  class CTreeItemT  {  public:  HTREEITEM m\_hTreeItem;  CTreeViewCtrlExT<TBase>\* m\_pTreeView;  ...  ...  typedef CTreeItemT<ATL::CWindow> CTreeItem; |

　CTreeViewCtrlEx 클래스는 CTreeViewCtrl클래스와 동등의 기능을 갖추고 있습니다. 이하에 나타내는 것은 CTreeViewCtrlEx 클래스를 사용하는 예 입니다. 전회 트리 뷰 컨트롤을 사용한 프로그램의 CTreeViewCtrl을 CTreeViewCtrlEx로 변경해 고쳐 씁니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  enum {SHELL = 1, COMCTL, WINDIR, SYSDIR}; // 트리 아이템 식별자    CTreeViewCtrlEx m\_tree\_sysinfo; // 확장 트리뷰  CStatic m\_static\_display; // 정보 표시용  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_HANDLER\_EX(IDC\_TREE\_SYSINFO, NM\_CLICK, OnTreeClick)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_tree\_sysinfo = GetDlgItem(IDC\_TREE\_SYSINFO);  m\_static\_display = GetDlgItem(IDC\_STATIC\_DISPLAY);  CTreeItem tiSysInfo =  m\_tree\_sysinfo.InsertItem(\_T("시스템 정보"), TVI\_ROOT, TVI\_LAST);  CTreeItem tiDllVer = tiSysInfo.AddTail(\_T("DLL버젼"), -1);  CTreeItem tiItem;  tiItem = tiDllVer.AddTail(\_T("shell32.dll"), -1);  tiItem.SetData(SHELL);  tiItem = tiDllVer.AddTail(\_T("comctl32.dll"), -1);  tiItem.SetData(COMCTL);  CTreeItem tiSysDir = tiSysInfo.AddTail(\_T("디렉토리"), -1);  tiItem = tiSysDir.AddTail(\_T("Windows"), -1);  tiItem.SetData(WINDIR);  tiItem = tiSysDir.AddTail(\_T("시스템"), -1);  tiItem.SetData(SYSDIR);  return TRUE;  }  LRESULT OnTreeClick(LPNMHDR pnmh){  // 클릭된 아이템 핸들 취득  UINT uFlag;  CPoint pt = ::GetMessagePos();  m\_tree\_sysinfo.ScreenToClient(&pt);  CTreeItem tiItem = m\_tree\_sysinfo.HitTest(pt, &uFlag);  if(tiItem.IsNull() || !(uFlag & TVHT\_ONITEM))  return 0;  // 아이템 데이터 취득  CString strInfo;  DWORD dwData = tiItem.GetData();  switch(dwData){  case SHELL:  {  // shell32.dll의 버젼 취득  DWORD dwMajor, dwMinor;  AtlGetShellVersion(&dwMajor, &dwMinor);  strInfo.Format(\_T("Version %d.%02d"), dwMajor, dwMinor);  break;  }  case COMCTL:  {  // comctl32.dll의 버젼 취득  DWORD dwMajor, dwMinor;  AtlGetCommCtrlVersion(&dwMajor, &dwMinor);  strInfo.Format(\_T("Version %d.%02d"), dwMajor, dwMinor);  break;  }  case WINDIR:  {  // Windows폴더 패스 취득  TCHAR szPath[MAX\_PATH];  GetWindowsDirectory(szPath, MAX\_PATH);  strInfo = szPath;  break;  }  case SYSDIR:  {  // 시스템 폴더 패스 취득  TCHAR szPath[MAX\_PATH];  GetSystemDirectory(szPath, MAX\_PATH);  strInfo = szPath;  break;  }  default:  break;  }  m\_static\_display.SetWindowText(strInfo);  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

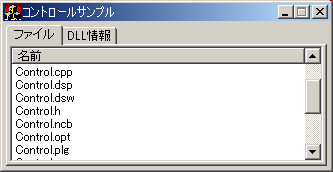
|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

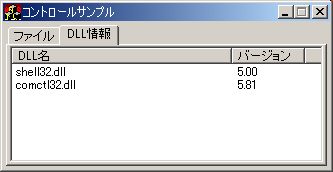
　우선 CMainDlg 클래스의 CTreeViewCtrl을 CTreeViewCtrlEx에 옮겨놓습니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러로 트리 뷰 컨트롤에 아이템을 삽입합니다만 전회는 InsertItem()의 리턴 값은 트리 아이템 핸들인 것에 대해 이번은CTreeViewCtrlEx 클래스를 사용하고 있으므로 리턴 값은 CTreeItem 클래스의 인스턴스입니다. 이 인스턴스 안에는 이미 트리 아이템 핸들과 m\_tree\_sysinfo 에의 포인터가 설정되어 있습니다. 삽입한 아이템 아래에 한층 더 아이 아이템을 추가하려면 CTreeItem의 인스턴스의 AddTail()을 호출합니다.   
  
　OnTreeClick()에서는 HitTest()의 리턴 값으로부터 CTreeItem 클래스의 인스턴스를 취득한 후 IsNull()에 의해 트리 아이템 핸들인가 유효한지 어떤지를 판단해서 유효한 경우는 GetData()에 의해 그 아이템으로부터 데이터를 취득하고 있습니다.   
  
이와 같이 ,CTreeViewCtrlEx클래스를 사용하면(자) , 특정의 트리 아이템을 기점으로서 트리뷰콘트로르를 조작할 수 있습니다.

**▣ 공용 컨트룰러 > 탭**  
WTL에서 공용 컨트롤의 탭 컨트롤을 CTabCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CTabCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CTabCtrlT<ATL::CWindow> CTabCtrl; |

이것은 CTabCtrl 클래스는 CWindow클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CTabCtrl클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CTabCtrl 클래스를 사용하는 예입니다. 첫째의 탭에는 현재 디렉토리의 파일 일람을 표시하는 리스트 뷰 컨트롤을 설정해 둘째의 탭에는DLL의 버젼 정보를 표시하는 리스트 뷰 컨트롤을 설정합니다.





|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CTabCtrl m\_tab\_display; // 탭  CListViewCtrl m\_list\_file; // [파일]리스트뷰  CListViewCtrl m\_list\_info; // [DLL정보]리스트뷰  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_HANDLER\_EX(IDC\_TAB\_DISPLAY, TCN\_SELCHANGING, OnTabSelChanging)  NOTIFY\_HANDLER\_EX(IDC\_TAB\_DISPLAY, TCN\_SELCHANGE, OnTabSelChange)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_tab\_display = GetDlgItem(IDC\_TAB\_DISPLAY);  m\_list\_file = GetDlgItem(IDC\_LIST\_FILE);  m\_list\_info = GetDlgItem(IDC\_LIST\_INFO);  // 탭 콘트롤의 배치  CRect rcClient;  GetClientRect(rcClient);  int nPaddingX = GetSystemMetrics(SM\_CXDLGFRAME);  int nPaddingY = GetSystemMetrics(SM\_CYDLGFRAME);  m\_tab\_display.SetWindowPos(HWND\_BOTTOM,  rcClient.left + nPaddingX, rcClient.top + nPaddingY,  rcClient.Width() - nPaddingX \* 2, rcClient.Height() - nPaddingY \* 2,  SWP\_SHOWWINDOW);  // 탭 아이템 삽입  TCITEM ti;  ti.mask = TCIF\_TEXT | TCIF\_PARAM;  ti.pszText = \_T("파일");  ti.lParam = (LPARAM)m\_list\_file.m\_hWnd;  m\_tab\_display.InsertItem(0, &ti);  ti.pszText = \_T("DLL정보");  ti.lParam = (LPARAM)m\_list\_info.m\_hWnd;  m\_tab\_display.InsertItem(1, &ti);  // 탭 콘트롤의 표시 영역 취득  CRect rcTabClient;  m\_tab\_display.GetClientRect(rcTabClient);  m\_tab\_display.AdjustRect(FALSE, rcTabClient);  rcTabClient.OffsetRect(nPaddingX, nPaddingY);  // [파일]리스트뷰콘트로르의 배치  m\_list\_file.MoveWindow(rcTabClient);  m\_list\_file.SetExtendedListViewStyle(LVS\_EX\_INFOTIP | LVS\_EX\_FULLROWSELECT);  CRect rcList;  m\_list\_file.GetWindowRect(rcList);  int nScrollWidth = GetSystemMetrics(SM\_CXVSCROLL);  int n3DEdge = GetSystemMetrics(SM\_CXEDGE);  m\_list\_file.InsertColumn(0, \_T("이름"), LVCFMT\_LEFT,  rcList.Width() - nScrollWidth - n3DEdge \* 2, -1);  CFindFile find;  if(find.FindFile()){  do{  if(!find.IsDots() && !find.IsDirectory()){  int nIndex = m\_list\_file.GetItemCount();  m\_list\_file.AddItem(nIndex, 0, find.GetFileName());  }  }while(find.FindNextFile());  }  // [DLL정보]리스트뷰콘트로르의 배치  m\_list\_info.MoveWindow(rcTabClient);  m\_list\_info.SetExtendedListViewStyle(LVS\_EX\_INFOTIP | LVS\_EX\_FULLROWSELECT);  m\_list\_info.InsertColumn(0, \_T("DLL명"), LVCFMT\_LEFT, 220, -1);  m\_list\_info.InsertColumn(1, \_T("버젼"), LVCFMT\_LEFT,  rcList.Width() - 220 - nScrollWidth - n3DEdge \* 2, -1);  DWORD dwMajor, dwMinor;  CString strInfo;  AtlGetShellVersion(&dwMajor, &dwMinor);  strInfo.Format(\_T("%d.%02d"), dwMajor, dwMinor);  m\_list\_info.AddItem(0, 0, \_T("shell32.dll"));  m\_list\_info.AddItem(0, 1, strInfo);  AtlGetCommCtrlVersion(&dwMajor, &dwMinor);  strInfo.Format(\_T("%d.%02d"), dwMajor, dwMinor);  m\_list\_info.AddItem(1, 0, \_T("comctl32.dll"));  m\_list\_info.AddItem(1, 1, strInfo);  return TRUE;  }  void SwitchWindow(bool bShow){  int nIndex = m\_tab\_display.GetCurSel();  TCITEM ti;  ti.mask = TCIF\_PARAM;  m\_tab\_display.GetItem(nIndex, &ti);  CWindow win = (HWND)ti.lParam;  win.ShowWindow(bShow ? SW\_SHOW : SW\_HIDE);  if(bShow)  win.SetFocus();  }  LRESULT OnTabSelChanging(LPNMHDR pnmh){  SwitchWindow(false);  return FALSE;  }  LRESULT OnTabSelChange(LPNMHDR pnmh){  SwitchWindow(true);  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 자원을 작성합니다. 다이얼로그에 탭 컨트롤과 리스트 뷰 컨트롤을 배치해 자원ID를 다음과 같이 지정합니다. 덧붙여 2개의 리스트 뷰 컨트롤의 [스타일] 에서는 디폴트에 [표시]로 [리포트]를 선택하고 [선택을 항상 표시]에 체크를 넣습니다. 게다가 [DLL정보]용 리스트 뷰 컨트롤만 [일반]의 [가시]의 체크를 뗍니다.

|  |  |
| --- | --- |
| 컨트롤명 | 자원ID |
| 탭 | IDC\_TAB\_DISPLAY |
| 리스트뷰([파일]용) | IDC\_LIST\_FILE |
| 리스트뷰([DLL정보]용) | IDC\_LIST\_INFO |

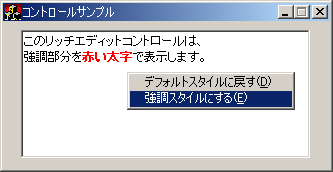
　다음에 stdafx.h 안에서는 CTabCtrl 클래스나 CListViewCtrl 클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   
  
　CMainDlg클래스에서는 탭 컨트롤 용으로 CTabCtrl클래스 리스트 뷰 컨트롤 용으로 CListViewCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것들을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러로 각 컨트롤을 배치합니다. 우선 탭 컨트롤을 다이얼로그보다 nPaddingX 와 nPaddingY의 분(margin) 만큼 작게 배치합니다. 그리고 탭에 2개의 아이템([파일]과[DLL정보])를 삽입합니다. 이 때 2개의 리스트 뷰 컨트롤의 핸들을 각각 TCITEM 구조체의 lParam 으로 설정합니다.   
  
그 후 , 탭 콘트롤의 표시 영역(탭 콘트롤의 탭 부분을 제외한 구형) 을 취득해 , 그 영역에 맞도록(듯이) 2개의 리스트뷰콘트로르를 배치합니다.   
  
　다음에 , 통지 코드가TCN\_SELCHANGING의 탭 콘트롤용 통지 메세지 핸들러를 추가합니다. 이 메세지 핸들러는 탭이 바뀌기 직전에 불려 갑니다만 , 여기에서는GetCurSel()에 의해 바뀌기 직전의 탭의 인덱스를 취득해 , 그 인덱스의 탭 아이템으로부터 리스트뷰콘트로르의 윈도우 핸들을 취득해 , 그 리스트뷰콘트로르를 불가시로 하고 있습니다.   
  
　다음에 , 통지 코드가TCN\_SELCHANGE의 탭 콘트롤용 통지 메세지 핸들러를 추가합니다. 이 메세지 핸들러는 탭이 바뀐 직후에 불려 갑니다만 , 여기에서는GetCurSel()에 의해 바뀐 직후의 탭의 인덱스를 취득해 , 그 인덱스의 탭 아이템으로부터 리스트뷰콘트로르의 윈도우 핸들을 취득해 , 그 리스트뷰콘트로르를 가시로 하고 있습니다.

▣ 공용 컨트룰러 > 리치 에디터

**< 리치 에디트 >**  
  
WTL에서는 공용 컨트롤의 리치 에디트 컨트롤을 CRichEditCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CRichEditCtrlT클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CRichEditCtrlT<ATL::CWindow> CRichEditCtrl; |

이것은 CRichEditCtrl 클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CRichEditCtrl클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있는 것 외에 리치 에디트 컨트롤의 버젼2.0과 3.0용의 멤버 함수도 갖추고 있습니다. 이하에 나타내는 것은 CRichEditCtrl클래스를 사용하는 예 입니다. 리치 에디트 컨트롤상에서 오른쪽 클릭하면 pop-up menu가 표시되어 [강조 스타일로 하는]을 선택하면 리치 에디트 컨트롤 상의 선택 부분이 붉은 굵은 글씨가 됩니다.



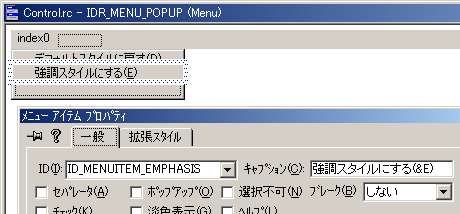
|  |
| --- |
| // stdafx.h  // 리치 에디트 컨트롤의 버젼1.0을 사용하는 것을 정의  #define \_RICHEDIT\_VER 0x0100  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CRichEditCtrl m\_rich\_emphasis; // 리치 에디트  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_DEFAULT, OnMenuDefault)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EMPHASIS, OnMenuEmphasis)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_rich\_emphasis = GetDlgItem(IDC\_RICHEDIT\_EMPHASIS);  return TRUE;  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 선택 캐릭터 라인의 마지막에 설정  if(pt.x == -1 && pt.y == -1){  LONG start, end;  m\_rich\_emphasis.GetSel(start, end);  pt = m\_rich\_emphasis.PosFromChar(end);  m\_rich\_emphasis.ClientToScreen(&pt);  }  // 좌표가 리치 에디트 컨트롤내의 경우만 pop-up menu를 표시  CRect rc;  m\_rich\_emphasis.GetClientRect(&rc);  m\_rich\_emphasis.ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  void OnMenuDefault(UINT uNotifyCode, int nID, HWND hWndCtl){  CString strText;  if(m\_rich\_emphasis.GetSelText(strText) != 0){  CHARFORMAT cf = {sizeof(CHARFORMAT)};  m\_rich\_emphasis.GetDefaultCharFormat(cf);  m\_rich\_emphasis.SetSelectionCharFormat(cf);  }  }  void OnMenuEmphasis(UINT uNotifyCode, int nID, HWND hWndCtl){  CString strText;  if(m\_rich\_emphasis.GetSelText(strText) != 0){  CHARFORMAT cf = {sizeof(CHARFORMAT)};  cf.dwMask = CFM\_COLOR | CFM\_BOLD;  cf.dwEffects = CFE\_BOLD;  cf.crTextColor = RGB(255, 0, 0);  m\_rich\_emphasis.SetSelectionCharFormat(cf);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  // 리치 에디트 컨트롤 초기화  HINSTANCE hRich = LoadLibrary(CRichEditCtrl::GetLibraryName());  if(hRich == NULL){  AtlMessageBox(NULL, \_T("리치 에디트 컨트롤 초기화 실패"),  \_T("에러"), MB\_OK | MB\_ICONERROR);  return 0;  }  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  FreeLibrary(hRich);  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 리치 에디트 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 리치 에디트 컨트롤의 [스타일]에서는 디폴트에 [복수행], [수평 스크롤 바], [수직 스크롤 바], [수직 오토 스크롤], [개행을 허가]에 체크를 넣습니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스ID |
| 리치 에디트 | IDC\_RICHEDIT\_EMPHASIS |

게다가 pop-up menu를 위한 메뉴 자원을 프로젝트에 추가합니다. 메뉴 리소스 ID는 IDR\_MENU\_POPUP으로 하며 톱 레벨에 [index0]를 그 아래에 [디폴트 스타일에 되돌리는] 메뉴 아이템과 [강조 스타일로 하는] 메뉴 아이템을 추가합니다. 아이템에는 각각 ID\_MENUITEM\_DEFAULTID\_MENUITEM\_EMPHASIS라고 하는 ID를 설정합니다.   
  
  
  
　다음에 stdafx.h에서는 CRichEditCtrl 클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다. 그리고 이번 예 에서는 버젼1.0의 리치 에디트 컨트롤을 사용하므로 \_RICHEDIT\_VER 를 0x0100 에 정의합니다. 덧붙여 이것은 AppWizard을 사용하면 디폴트로 정의되고 있습니다.   
  
　CMainDlg클래스에서는 리치 에디트 컨트롤용으로 CRichEditCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　다음에 WM\_CONTEXTMENU 메세지 핸들러를 추가합니다. 여기에서는 좌표가 리치 에디트 컨트롤 내인지 어떤지를 확인해 pop-up menu를 표시하고 있습니다.   
  
　다음에 pop-up menu의 아이템을 선택했을 때의 커멘드 메세지 핸들러를 추가합니다. [디폴트 스타일에 되돌리는] (ID：ID\_MENUITEM\_DEFAULT) 의 커멘드 메세지 핸들러 OnMenuDefault()에서는 GetDefaultCharFormat()에 의해 디폴트의 문자 포맷을 취득해 SetSelectionCharFormat()에 의해 선택 부분에 그 포맷을 적용하고 있습니다.   
  
[강조 스타일로 하는](ID：ID\_MENUITEM\_EMPHASIS) 의 커멘드 메세지 핸들러OnMenuEmphasis()에서는 , 붉은 굵은 글씨를 표시하기 위한CHARFORMAT구조체를 작성해 , SetSelectionCharFormat()에 의해 선택 부분에 그 포맷을 적용하고 있습니다.   
  
마지막으로 \_tWinMain()에서 리치 에디트 컨트롤을 초기화합니다. CRichEditCtrl::GetLibraryName()는 \_RICHEDIT\_VER 이 0x0100의 경우는\_T("RICHED32.DLL")를 리턴하며 0x0200 이상의 경우는\_T("RICHED20.DLL")을 리턴 합니다.   
  
　이와 같이 리치 에디트 컨트롤을 사용하는 경우는 \_RICHEDIT\_VER의 값에 사용하는 에디트 컨트롤의 버젼을 지정해 LoadLibrary()를 호출하는 것에 의해 초기화를 할 필요가 있습니다.

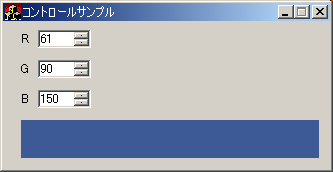
**▣ 공용 컨트룰러 > 트랙바**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WTL에서 공용 컨트롤의 트랙 바(또는 슬라이더) 컨트롤을 CTrackBarCtrlT 이라고 하는 템플릿 클래스에서 캡슐화 하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CTrackBarCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlctrls.h  typedef CTrackBarCtrlT<ATL::CWindow> CTrackBarCtrl; |   이것은 CTrackBarCtrl 클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   　WTL의 CTrackBarCtrl클래스는 MFC의 CSliderCtrl클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CTrackBarCtrl 클래스를 사용하는 예 입니다. 트럭 바를 왼쪽으로 이동하는 만큼 다이얼로그의 투명도를 높게 해 표시합니다. 덧붙여 이 프로그램은 반투명 기능(SetLayeredWindowAttributes()) 을 사용하기 때문에 Windows 2000이후 버전에 실행할 필요가 있습니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl10-10/01.png   |  | | --- | | // stdafx.h  #define \_WIN32\_WINNT 0x0500  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CTrackBarCtrl m\_track\_layer; // 트랙 바  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  MSG\_WM\_HSCROLL(OnHScroll)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 확장 윈도우 스타일 설정  ModifyStyleEx(0, WS\_EX\_LAYERED);  // 컨트롤 설정  m\_track\_layer = GetDlgItem(IDC\_SLIDER\_LAYER);  m\_track\_layer.SetRange(0, 255);  m\_track\_layer.SetTicFreq(15);  m\_track\_layer.SetPageSize(15);  m\_track\_layer.SetLineSize(1);  m\_track\_layer.SetPos(255);  SetLayer(m\_track\_layer.GetPos());  return TRUE;  }  void OnHScroll(int nCode, short nPos, HWND hWnd){  CTrackBarCtrl trackbar = hWnd;  SetLayer(trackbar.GetPos());  }  void SetLayer(BYTE byteAlpha){  HMODULE hDll = LoadLibrary(\_T("user32"));  if(hDll){  typedef BOOL (WINAPI \*PSLWA) (HWND, COLORREF, BYTE, DWORD);  PSLWA pSetLayeredWindowAttributes  = (PSLWA)GetProcAddress(hDll, "SetLayeredWindowAttributes");  if(pSetLayeredWindowAttributes != NULL)  pSetLayeredWindowAttributes(m\_hWnd, 0, byteAlpha, LWA\_ALPHA);  FreeLibrary(hDll);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  // Windows의 버젼 체크  OSVERSIONINFO vi = {sizeof(OSVERSIONINFO)};  GetVersionEx(&vi);  if(vi.dwPlatformId != VER\_PLATFORM\_WIN32\_NT || vi.dwMajorVersion < 5){  AtlMessageBox(NULL, \_T("OS가 반투명 기능에 대응하고 있지 않습니다. "),  \_T("에러"), MB\_OK | MB\_ICONERROR);  return 0;  }  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 리소스를 작성합니다. 다이얼로그에 트랙 바(슬라이더) 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 트랙 바 컨트롤의 [스타일]에서는 디폴트에 [마크의 설정]과 [자동 설정]에 체크를 넣습니다.   |  |  | | --- | --- | | 컨트롤명 | 리소스ID | | 트럭 바 | IDC\_SLIDER\_LAYER |   　다음에 stdafx.h에서는 CTrackBarCtrl클래스를 사용하기 위해서 atlctrls.h헤더를 인클루드 합니다. 그리고 이번 예에서는SetLayeredWindowAttributes()API을 사용하므로 \_WIN32\_WINNT 를 0x0500 에 정의합니다.   　CMainDlg클래스에서는 우선 트럭 바 컨트롤용으로 CTrackBarCtrl클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   　다음에 WM\_INITDIALOG 메세지 핸들러로 확장 윈도우 스타일에 WS\_EX\_LAYERED를 추가해 그 후 트랙 바 컨트롤의 초기설정을 합니다. 여기에서는 범위(최소치 0, 최대치255) , 눈금 마크의 잘게 썰기(15마다 눈금 마크를 표시) 페이지 사이즈([PageUp][PageDown]키를 눌렀을 때에15두개 이동) 라인 사이즈(커서 키를 눌렀을 때에 1 두개 이동) 를 설정해 있습니다. 게다가 안주의 초기 위치를255로 하여 그 위치를SetLayer()에 건네주고 있습니다. 이것은 디폴트로 불투명한 다이얼로그를 표시하는 것을 의미합니다.   　다음에 WM\_HSCROLL메세지 핸들러를 추가합니다. 트랙 바 컨트롤은 스크롤 바와 같이 바를 이동하면 WM\_HSCROLL메세지를 부모 윈도우에 보냅니다. WM\_HSCROLL메세지 핸들러에서는 트랙 바 컨트롤의 현재 위치를 취득해 SetLayer()함수에 건네주고 있습니다.   SetLayer()함수에서는 ,user32.dll으로부터 SetLayeredWindowAttributes()API의 주소를 취득해 , 다이얼로그를 투명도를 변경하고 있습니다.   　마지막으로 ,\_tWinMain()안에서GetVersionEx()를 호출하는 것에 의해Windows의 버젼 체크를 합니다. |

**▣ 공용 컨트룰러 > 업 다운**  
WTL에서 공용 컨트롤의 updown(또는 스핀 버튼) 컨트롤을 CUpDownCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CUpDownCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CUpDownCtrlT<ATL::CWindow> CUpDownCtrl; |

이것은 CUpDownCtrl클래스는CWindow클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CUpDownCtrl클래스는 MFC의 CSpinButtonCtrl클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CUpDownCtrl클래스를 사용하는 예 입니다. RGB 각각의 updown 컨트롤을 조작해 색을 작성합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CUpDownCtrl m\_updown\_r; // [R]용  CUpDownCtrl m\_updown\_g; // [G]용  CUpDownCtrl m\_updown\_b; // [B]용  CRect m\_rect; // 색 표시용 구형  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_HANDLER\_EX(IDC\_EDIT\_R, EN\_CHANGE, OnEditChangeR)  COMMAND\_HANDLER\_EX(IDC\_EDIT\_G, EN\_CHANGE, OnEditChangeG)  COMMAND\_HANDLER\_EX(IDC\_EDIT\_B, EN\_CHANGE, OnEditChangeB)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_updown\_r = GetDlgItem(IDC\_SPIN\_R);  m\_updown\_g = GetDlgItem(IDC\_SPIN\_G);  m\_updown\_b = GetDlgItem(IDC\_SPIN\_B);  m\_updown\_r.SetRange(0, 255);  m\_updown\_g.SetRange(0, 255);  m\_updown\_b.SetRange(0, 255);  // 색 표시용 구형을 취득  CStatic view = GetDlgItem(IDC\_STATIC\_VIEW);  view.GetWindowRect(m\_rect);  ScreenToClient(m\_rect);  return TRUE;  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CBrush brush;  brush.CreateSolidBrush(RGB(m\_updown\_r.GetPos(),  m\_updown\_g.GetPos(), m\_updown\_b.GetPos()));  dc.FillRect(m\_rect, brush);  }  void CheckAndUpdate(CUpDownCtrl& updown){  if(updown.m\_hWnd != NULL){  // 범위 체크  BOOL bError;  updown.GetPos(&bError);  if(bError)  updown.SetPos(255);  }  InvalidateRect(m\_rect);  }  void OnEditChangeR(UINT uNotifyCode, int nID, HWND hWndCtl){  CheckAndUpdate(m\_updown\_r);  }  void OnEditChangeG(UINT uNotifyCode, int nID, HWND hWndCtl){  CheckAndUpdate(m\_updown\_g);  }  void OnEditChangeB(UINT uNotifyCode, int nID, HWND hWndCtl){  CheckAndUpdate(m\_updown\_b);  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 자원을 작성합니다. 다이얼로그에 스태틱, 에디트, updown 컨트롤을 배치하고 리소스 ID를 다음과 같이 지정합니다. 덧붙여 에디트 컨트롤의 [스타일]에서는 디폴트에 [번호]에 체크를 넣어 updown 컨트롤의 [스타일]에서는 디폴트에 [배치]로 [우단]을 선택하고 [자동 관련짓고], [수치의 자동 표시], [자리수 단락지어 이루어], [워드랩]에 체크를 넣습니다. 또한 색 표시용의 스태틱 컨트롤의 프로퍼티에서는 [가시]의 체크를 off 합니다.

|  |  |
| --- | --- |
| 컨트롤 명 | 리소스ID |
| 스태틱([R]라벨용) | IDC\_STATIC(디폴트) |
| 스태틱([G]라벨용) | IDC\_STATIC(디폴트) |
| 스태틱([B]라벨용) | IDC\_STATIC(디폴트) |
| 에디트([R]용) | IDC\_EDIT\_R |
| 에디트([G]용) | IDC\_EDIT\_G |
| 에디트([B]용) | IDC\_EDIT\_B |
| updown([R]용) | IDC\_SPIN\_R |
| updown([G]용) | IDC\_SPIN\_G |
| updown([B]용) | IDC\_SPIN\_B |
| 스태틱(색 표시용) | IDC\_STATIC\_VIEW |

updown 컨트롤의 [자동 관련짓고]에 체크 하는 것으로 Z오더를 기본으로 updown 컨트롤의 직전의 컨트롤에 자동적으로 관련 지을 수 있습니다. 이번 예에서는 [R]용 에디트 컨트롤 다음에 [R]용 updown 컨트롤을 배치하는 것으로 [R]용 에디트 컨트롤과 [R]용 updown 컨트롤을 관련 지을 수 있습니다. 같이 [G]나 [B] 의 컨트롤도 차례로 배치합니다.   
  
　다음에 stdafx.h 에서는 CUpDownCtrl 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   
  
　CMainDlg클래스에서는 우선 updown 컨트롤용으로 CUpDownCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다. 그리고 색을 표시하는 구형을 나타내는 CRect 클래스의 인스턴스도 멤버 변수로서 선언해 둡니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러로 updown 컨트롤의 범위를 설정합니다. 그리고 색 표시용의 스태틱 구형을 취득합니다.   
  
　다음에 통지 코드가 EN\_CHANGE의 에디트 컨트롤용 커멘드 메세지 핸들러를 [R][G][B] 각각 추가합니다. 이 핸들러는 에디트 컨트롤의 내용이 변경되었을 때 (직접치를 입력했을 때와 updown 컨트롤로 값을 변경했을 때 ) 에 호출 되어 집니다. 이 핸들러에서는 CheckAndUpdate() 이라고 하는 독자적인 함수를 호출해 GetPos()에 의해 값의 범위가 정당한지 어떤지 체크하고 나서 색 표시용 구형을 갱신합니다.   
  
　마지막으로 WM\_PAINT 메세지 핸들러를 추가해 색을 묘화 합니다.

▣ 공용 컨트룰러 > 프로그레스바

**< Progress bar >**WTL에서 코먼 컨트롤의 progress bar 컨트롤을 CProgressBarCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CProgressBarCtrlT 클래스의 base class 로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CProgressBarCtrlT<ATL::CWindow> CProgressBarCtrl; |

이것은 CProgressBarCtrl클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CProgressBarCtrl클래스는 MFC의 CProgressCtrl클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CProgressBarCtrl클래스를 사용하는 예입니다. 100밀리 세컨드 마다 바를 진행시킵니다. 끝까지 진행되는데 약 1초 걸립니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CProgressBarCtrl m\_progress\_sec;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_START, OnButtonStart)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_progress\_sec = GetDlgItem(IDC\_PROGRESS\_SEC);  m\_progress\_sec.SetRange(0, 10);  m\_progress\_sec.SetPos(0);  return TRUE;  }  void OnButtonStart(UINT uNotifyCode, int nID, HWND hWndCtl){  for(int i=0; i<=10; i++){  m\_progress\_sec.SetPos(i);  Sleep(100);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 progress bar 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 progress bar 컨트롤의 [스타일]에서는 디폴트에 [스무드]에 체크를 넣습니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스ID |
| progress bar | IDC\_PROGRESS\_SEC |

　다음에 stdafx.h에서는 CProgressBarCtrl클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   
  
　CMainDlg 클래스에서는 우선 progress bar 컨트롤 용으로 CProgressBarCtrl클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　다음으로 WM\_INITDIALOG 메세지 핸들러로 progress bar 컨트롤의 범위와 초기 위치를 설정합니다.   
　 버튼 컨트롤용 커멘드 메세지 핸들러를 추가합니다. 그 핸들러에서는SetPos()에 의해 바를 1/10두 개 진행합니다. 덧붙여 Sleep()에 의해100밀리 세컨드마다 진행하기 때문에 끝까지 진행되는데 약 1초 걸립니다.   
  
　위의 예에서는 바를 진행시키기 위해서(때문에)SetPos()를 사용했습니다만 , progress bar 컨트롤은 , 바를 진행시키기 위해서(때문에) 다른 2개의 방법을 준비해 있습니다.   
  
　첫째의 방법은 OffsetPos()을 사용하는 방법입니다. OffsetPos()는 바의 위치를 현재의 위치에 대해서 상대적인 위치로 설정합니다.

|  |
| --- |
| for(int i=0; i<10; i++) {  m\_progress\_sec.OffsetPos(1); // 현재의 위치에 대해서 1 진행한다  Sleep(100);  } |

　둘째의 방법은 StepIt()을 사용하는 방법입니다. StepIt()을 사용하는 경우는 미리 SetStep()로 증분치를 설정합니다. 그 후 StepIt()를 호출할 때마다 증분치의 분 만큼 바가 진행됩니다.

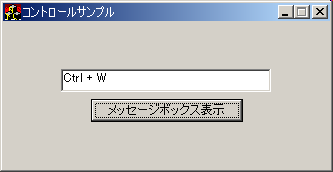
|  |
| --- |
| m\_progress\_sec.SetStep(1);  for(int i=0; i<10; i++){  m\_progress\_sec.StepIt();  Sleep(100);  } |

▣ 공용 컨트룰러 > 핫키

**< Hot Key >**WTL에서 공용 컨트롤의 hot key 컨트롤을 CHotKeyCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는CHotKeyCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CHotKeyCtrlT<ATL::CWindow> CHotKeyCtrl; |

이것은 CHotKeyCtrl클래스는CWindow 클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CHotKeyCtrl클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CHotKeyCtrl 클래스를 사용하는 예 입니다. hot key 컨트롤에 입력한 키의 편성에 의해 표시하는 메시지 박스의 내용을 변경합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CHotKeyCtrl m\_hotkey\_msgbox;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_MSGBOX, OnButtonMsgBox)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_hotkey\_msgbox = GetDlgItem(IDC\_HOTKEY\_MSGBOX);  m\_hotkey\_msgbox.SetHotKey(\_T('A'), HOTKEYF\_CONTROL);  return TRUE;  }  void OnButtonMsgBox(UINT uNotifyCode, int nID, HWND hWndCtl){  WORD wVirtualKeyCode, wModifiers;  m\_hotkey\_msgbox.GetHotKey(wVirtualKeyCode, wModifiers);  if(wVirtualKeyCode == \_T('A') && wModifiers == HOTKEYF\_CONTROL)  MessageBox(\_T("Hello, ATL"));  else if(wVirtualKeyCode == \_T('W') && wModifiers == HOTKEYF\_CONTROL)  MessageBox(\_T("Hello, WTL"));  else  MessageBox(\_T("메세지 없음"));  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 hot key 컨트롤과 버튼 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스ID |
| hot key | IDC\_HOTKEY\_MSGBOX |
| 버튼 | IDC\_BUTTON\_MSGBOX |

　다음에 stdafx.h 에서는 CHotKeyCtrl 클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   
  
　CMainDlg클래스에서는 우선 hot key용으로 CHotKeyCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　WM\_INITDIALOG메세지 핸들러로 hot key에 초기 상태로서 [Ctrl]+[A] 키를 설정합니다.   
  
　버튼 컨트롤용 커멘드 메세지 핸들러를 추가합니다. 그 핸들러에서는 GetHotKey()에 의해 입력된 키코드를 취득해 그 키에 의해 표시하는 메시지 박스를 바꾸고 있습니다. hot key 컨트롤에 [Ctrl]+[A]를 입력했을 경우는 「Hello, ATL」, [Ctrl]+[W]을 입력했을 경우는 「Hello, WTL」, 그 이외의 경우는 「메세지 없음」이라고 하는 메시지 박스를 표시합니다.

▣ 공용 컨트룰러 > 애니메이션

**< 애니메이션 >**WTL에서 공용 컨트롤의 애니메이션 컨트롤을 CAnimateCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는CAnimateCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CAnimateCtrlT<ATL::CWindow> CAnimateCtrl; |

이것은 CAnimateCtrl클래스는 CWindow클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CAnimateCtrl클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CAnimateCtrl클래스를 사용하는 예입니다. 현재 디렉토리에 준비한 filecopy.avi라고 하는 파일을 재생합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h안  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CAnimateCtrl m\_anime\_filecopy;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_PLAY, OnButtonPlay)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_STOP, OnButtonStop)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_anime\_filecopy = GetDlgItem(IDC\_ANIMATE\_FILECOPY);  m\_anime\_filecopy.Open(\_T("filecopy.avi"));  return TRUE;  }  void OnButtonPlay(UINT uNotifyCode, int nID, HWND hWndCtl){  m\_anime\_filecopy.Play(0, -1, -1);  }  void OnButtonStop(UINT uNotifyCode, int nID, HWND hWndCtl){  m\_anime\_filecopy.Stop();  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  m\_anime\_filecopy.Close();  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  m\_anime\_filecopy.Close();  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 자원을 작성합니다. 다이얼로그에 애니메이션 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 애니메이션 컨트롤의 [스타일]에서는 디폴트에 [투과]와 [중앙]에 체크를 넣어 [경계선]의 체크를 뗍니다.

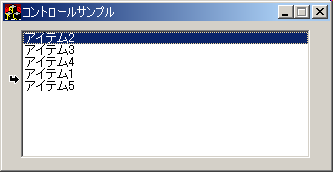
|  |  |
| --- | --- |
| 컨트롤명 | 리소스ID |
| 애니메이션 | IDC\_ANIMATE\_FILECOPY |

　 stdafx.h 에서는 CAnimateCtrl 클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   
  
　CMainDlg클래스에서는 우선 애니메이션 컨트롤 용으로 CAnimateCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　 WM\_INITDIALOG 메세지 핸들러로, 현재 디렉토리에 있는 filecopy.avi 라고 하는 파일을 엽니다.   
  
　 [재생]과 [정지]의 버튼 컨트롤용 커멘드 메세지 핸들러를 추가합니다. 그 핸들러에서는 각각 Play(), Stop()을 호출해 AVI 파일의 재생과 정지를 합니다. 덧붙여 Play()의 제1 인수에는 재생하는 AVI 파일의 시작 프레임 인덱스, 제2 인수에는 종료 프레임 인덱스( -1를 지정했을 경우는 자동적으로 마지막 프레임), 제3 인수에는 반복하는 회수( -1를 지정했을 경우는 제한 없음) 를 지정합니다.   
  
　마지막으로 ,OnOK라고OnCancel로Close()를 호출해 , 다이얼로그를 다물기 전에AVI파일을 닫습니다.

**▣ 공용 컨트룰러 > 드래그 리스트 박스**  
WTL는 리스트 박스 컨트롤에 드러그＆드롭에 의해 아이템의 순서를 변경하는 기능을 추가한 드래그 리스트 박스 컨트롤을 작성하기 위해서 CDragListBoxT 라는 CDragListNotifyImpl라고 하는 템플릿 클래스를 준비해 있습니다. CDragListBoxT 클래스는 CListBoxT 클래스의 상속 클래스입니다.   
atlctrls.h 헤더에서는 다음과 같이 정의되고 있습니다.

|  |
| --- |
| // atlctrls.h  template <class TBase>  class CDragListBoxT : public CListBoxT< TBase >  {  ...  ...  };  typedef CDragListBoxT<ATL::CWindow> CDragListBox; |

　WTL의 CDragListBox 클래스와 CDragListNotifyImpl클래스를 사용하는 것으로 MFC의 CDragListBox 클래스와 동등의 기능을 실현할 수 있습니다. 이하에 나타내는 것은 CDragListBox 클래스와 CDragListNotifyImpl 클래스를 사용해 드래그 리스트 박스를 작성하는 예 입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>, public CDragListNotifyImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CDragListBox m\_list\_drag;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  CHAIN\_MSG\_MAP(CDragListNotifyImpl<CMainDlg>) // base class에 체인  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_list\_drag = GetDlgItem(IDC\_LIST\_DRAG);  m\_list\_drag.MakeDragList();  m\_list\_drag.AddString(\_T("아이템1"));  m\_list\_drag.AddString(\_T("아이템2"));  m\_list\_drag.AddString(\_T("아이템3"));  m\_list\_drag.AddString(\_T("아이템4"));  m\_list\_drag.AddString(\_T("아이템5"));  return TRUE;  }  BOOL OnBeginDrag(int nCtlID, HWND hWndDragList, POINT ptCursor){  // 커서 위치의 아이템의 좌측으로 삽입 마크를 묘화  m\_list\_drag.DrawInsert(m\_list\_drag.LBItemFromPt(ptCursor));  return TRUE;  }  void OnCancelDrag(int nCtlID, HWND hWndDragList, POINT ptCursor){  // 삽입 마크를 소거  m\_list\_drag.DrawInsert(-1);  }  int OnDragging(int nCtlID, HWND hWndDragList, POINT ptCursor){  // 커서 위치의 아이템의 좌측으로 삽입 마크를 묘화  m\_list\_drag.DrawInsert(m\_list\_drag.LBItemFromPt(ptCursor));  return 0;  }  void OnDropped(int nCtlID, HWND hWndDragList, POINT ptCursor){  // 삽입 마크를 소거  m\_list\_drag.DrawInsert(-1);  int nSrcIndex = m\_list\_drag.GetCurSel(); // 이동원의 인덱스  int nDestIndex = m\_list\_drag.LBItemFromPt(ptCursor); // 이동처의 인덱스  if(nSrcIndex == -1 || nDestIndex == -1 || nDestIndex == nSrcIndex)  return;  CString strText;  m\_list\_drag.GetText(nSrcIndex, strText);  m\_list\_drag.DeleteString(nSrcIndex);  m\_list\_drag.InsertString(nDestIndex, strText);  m\_list\_drag.SetCurSel(nDestIndex);  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 리스트 박스 컨트롤을 배치하고 리소스 ID를 다음과 같이 지정합니다. 덧붙여 LBS\_MULTIPLESEL또는 LBS\_EXTENDEDSEL스타일은 지정 해서는 안됩니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 리스트 박스 | IDC\_LIST\_DRAG |

　다음으로 stdafx.h 에서는 CDragListBox 클래스와 CDragListNotifyImpl클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   
  
　CDragListNotifyImpl클래스는 메인 윈도우인 CMainDlg클래스의 base class로서 사용합니다. CMainDlg클래스에서는 우선 드래그 리스트 박스용으로 CDragListBox 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　다음으로 WM\_INITDIALOG 메세지 핸들러에서는 MakeDragList()을 호출해 리스트 박스를 드래그 리스트 박스화해 그 드래그 리스트 박스 컨트롤에 아이템을 추가합니다.   
  
　다음으로 메세지 맵에 base class인 CDragListNotifyImpl클래스에의 체인을 추가합니다. 이것에 의해 드레그 리스트 박스 컨트롤 용의 메세지가 등록되어 그 메세지가 메인 윈도우에 보내져 오면 체인에 의해 CDragListNotifyImpl클래스의 멤버 함수가 불려 갑니다. CDragListNotifyImpl클래스는 메세지의 통지 코드에 의해 다음과 같은 멤버 함수를 준비해 있습니다.

|  |  |
| --- | --- |
| 통지 코드 | 멤버 함수명 |
| DL\_BEGINDRAG | OnBeginDrag |
| DL\_CANCELDRAG | OnCancelDrag |
| DL\_DRAGGING | OnDragging |
| DL\_DROPPED | OnDropped |

CDragListNotifyImpl클래스의 상속 클래스는 이러한 멤버 함수를 오버라이드(override) 하는 것에 의해 드래그 리스트 박스 컨트롤 내의 아이템이 드래그＆드롭 되었을 때의 동작을 커스터마이즈 할 수가 있습니다.   
  
　위의 예 에서는 4개의 멤버 함수 모든 것을 오버라이드(override) 하고 있습니다. 우선 OnBeginDrag()은 아이템 드래그 하기 직전(마우스 커서로 선택했을 때 )에 OnDragging()는 드래그 한중간에 불려 집니다만 어느 쪽이나 DrawInsert()를 사용해 삽입 마크를 그리고 있습니다. OnDropped()은 드롭 되었을 때에 불려 갑니다만 여기에서는 우선DrawInsert()에 -1를 건네주어 삽입 마크를 소거해 다음에 선택되고 있는 아이템의 인덱스와 드롭 된 위치의 아이템의 인덱스를 취득합니다. 이러한 인덱스가 에러 또는 같은 경우 때는 굳이 하지 않고 끝납니다. 그렇지 않은 경우는 선택되고 있는 아이템 캐릭터 라인을 취득해 아이템을 삭제해 그 후에 드롭 된 위치에 그 아이템을 삽입합니다. 덧붙여OnCancelDrag()는 드래그를 중지 했을 때에 불려 집니다만 여기에서는 삽입 마크를 제거하고 있습니다.

▣ 공용 컨트룰러 > 확장 콤보 박스

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| WTL에서 공용 컨트롤의 확장 combobox 컨트롤을CComboBoxExT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. CComboBoxExT 클래스는 CComboBoxT클래스의 상속 클래스입니다. atlctrls.h 헤더에서는 다음과 같이 정의되고 있습니다.   |  | | --- | | // atlctrls.h  template <class TBase>  class CComboBoxExT : public CComboBoxT< TBase >  {  ...  ...  };  typedef CComboBoxExT<ATL::CWindow> CComboBoxEx; |   　WTL의 CComboBoxEx클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CComboBoxEx클래스를 사용하는 예입니다. 커런트 디렉토리의 파일명과 폴더 명을 일람표에 표시하여 파일명을 선택하면 파일 사이즈를 스태틱 컨트롤에 표시합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl10-16/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CComboBoxEx m\_combo\_mode; // 커런트 디렉토리 일람표시용  CStatic m\_static\_size; // 파일 사이즈 표시용  // 메시지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_HANDLER\_EX(IDC\_COMBOBOXEX\_DIR, CBN\_SELCHANGE, OnComboSelChange)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_combo\_mode = GetDlgItem(IDC\_COMBOBOXEX\_DIR);  m\_static\_size = GetDlgItem(IDC\_STATIC\_SIZE);  // 이미지 리스트를 작성  CImageList il;  il.Create(16, 16, ILC\_COLOR8 | ILC\_MASK, 2, 1);  CBitmap bmp;  bmp.LoadBitmap(IDB\_BITMAP\_ICON);  il.Add(bmp, RGB(255, 0, 255));  // 확장 combobox에 이미지 리스트를 설정  m\_combo\_mode.SetImageList(il);  // 확장 combobox에 아이템 추가  CFindFile find;  if(find.FindFile()){  do{  if(!find.IsDots())  InsertComboItem(find.GetFileName(), find.IsDirectory() ? 0 : 1);  }while(find.FindNextFile());  }  return TRUE;  }  void InsertComboItem(LPCTSTR lpszName, int nIcon){  COMBOBOXEXITEM cbei;  cbei.mask = CBEIF\_TEXT | CBEIF\_IMAGE | CBEIF\_SELECTEDIMAGE;  cbei.pszText = const\_cast<LPTSTR>(lpszName);  cbei.cchTextMax = lstrlen(lpszName);  cbei.iItem = m\_combo\_mode.GetCount();  cbei.iImage = nIcon;  cbei.iSelectedImage = nIcon;  m\_combo\_mode.InsertItem(&cbei);  }  void OnComboSelChange(UINT uNotifyCode, int nID, HWND hWndCtl){  int nIndex = m\_combo\_mode.GetCurSel();  if(nIndex != CB\_ERR){  // 선택된 아이템 캐릭터 라인 취득  CString strText;  m\_combo\_mode.GetLBText(nIndex, strText);  // 풀 패스를 작성  TCHAR szPath[\_MAX\_PATH];  LPTSTR lpFilePart;  GetFullPathName(strText,  sizeof(szPath)/sizeof(TCHAR), szPath, &lpFilePart);  // 파일 사이즈 취득  CString strMsg;  HANDLE hFile = CreateFile(szPath, 0, 0, NULL,  OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);  if(hFile != INVALID\_HANDLE\_VALUE){  DWORD dwSize = GetFileSize(hFile, NULL);  CloseHandle(hFile);  strMsg.Format(\_T("파일 사이즈: %ld아르바이트"), dwSize);  }  m\_static\_size.SetWindowText(strMsg);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES | ICC\_USEREX\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 리소스를 작성합니다. 다이얼로그에 확장 combobox 컨트롤과 스태틱 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 확장 combobox 컨트롤의 [스타일]에서는 [타입]으로 [드롭 다운 리스트]를 선택합니다.   |  |  | | --- | --- | | 컨트롤명 | 리소스ID | | 확장 combobox | IDC\_COMBOBOXEX\_DIR | | 스태틱(사이즈 표시용) | IDC\_STATIC\_SIZE |   　다음에 stdafx.h에서는 CComboBoxEx 클래스와 CStatic 클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   　CMainDlg클래스에서는 우선 확장 combobox 컨트롤과 스태틱 컨트롤 용으로 각각 CComboBoxEx 클래스와 CStatic 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것들을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   　다음에 WM\_INITDIALOG 메세지 핸들러로 이미지 리스트를 작성해 확장 combobox 컨트롤로 설정합니다. 그 후 확장 combobox 컨트롤에 현재 디렉토리의 파일명과 폴더명을 삽입합니다.   　다음에 통지 코드가 CBN\_SELCHANGE의 확장 combobox 컨트롤용 커멘드 메세지 핸들러를 추가합니다. 그 핸들러 에서는 현재 선택되고 있는 아이템을 취득해 그 파일 사이즈를 스태틱 컨트롤에 표시합니다.   　마지막으로 \_tWinMain() 안에서ICC\_USEREX\_CLASSES플래그를 지정해 AtlInitCommonControls()를 호출합니다. 이것에 의해 확장 combobox 컨트롤이 사용 가능하게 됩니다. 덧붙여 비교적 사용 빈도가 높다고 생각되는 ICC\_COOL\_CLASSES플래그와ICC\_WIN95\_CLASSES플래그도 동시에 지정해 있습니다만 이번 예에서는 이 두 개의 플래그는 없어도 괜찮습니다. |

**▣ 공용 컨트룰러 > 월간 달력**

WTL은 공용 컨트롤의 월간 캘린더 컨트롤을 CMonthCalendarCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CMonthCalendarCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CMonthCalendarCtrlT<ATL::CWindow> CMonthCalendarCtrl; |

이것은 CMonthCalendarCtrl클래스는 CWindow클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CMonthCalendarCtrl클래스는 MFC의 CMonthCalCtrl클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CMonthCalendarCtrl클래스를 사용하는 예입니다. 날짜를 선택하면 그 날짜를 메시지 박스로 표시합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CMonthCalendarCtrl m\_calendar\_getdate;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_HANDLER\_EX(IDC\_MONTHCALENDAR\_GETDATE, MCN\_SELECT, OnCalendarSelect)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_calendar\_getdate = GetDlgItem(IDC\_MONTHCALENDAR\_GETDATE);  CRect rcCalendar;  m\_calendar\_getdate.GetMinReqRect(rcCalendar);  m\_calendar\_getdate.MoveWindow(rcCalendar);  m\_calendar\_getdate.CenterWindow();  // 초기 상태의 일시 설정  SYSTEMTIME st = {0};  st.wYear = 2004;  st.wMonth = 2;  st.wDay = 7;  m\_calendar\_getdate.SetCurSel(&st);  return TRUE;  }  LRESULT OnCalendarSelect(LPNMHDR pnmh){  LPNMSELCHANGE pnsc = (LPNMSELCHANGE)pnmh;  CString strDate;  strDate.Format(\_T("%d연%d월%d일"),  pnsc->stSelStart.wYear,  pnsc->stSelStart.wMonth,  pnsc->stSelStart.wDay);  MessageBox(strDate);  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES | ICC\_DATE\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 월간 캘린더 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 월간 캘린더 | IDC\_MONTHCALENDAR\_GETDATE |

　다음에 stdafx.h 안에서는 CMonthCalendarCtrl 클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   
  
　CMainDlg클래스에서는 우선 월간 캘린더 컨트롤 용으로 CMonthCalendarCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　다음으로 WM\_INITDIALOG 메세지 핸들러로 월간 캘린더 컨트롤의 구형을 취득해 다이얼로그의 중앙에 표시합니다. 그리고 SetCurSel()에 의해 초기 상태의 일시를 2004년2월7일로 설정합니다.   
  
　다음으로 통지 코드가 MCN\_SELECT의 월간 캘린더 컨트롤용 통지 메세지 핸들러를 추가합니다. 그 핸들러에서는 선택된 아이템의 일자를 취득해 메시지 박스로 표시하고 있습니다.   
  
　마지막으로 \_tWinMain()안에서 ICC\_DATE\_CLASSES 플래그를 지정해 AtlInitCommonControls()를 호출합니다. 이것에 의해 월간 캘린더 컨트롤이 사용 가능하게 됩니다. 덧붙여 비교적 사용 빈도가 높다고 생각되는 ICC\_COOL\_CLASSES플래그와 ICC\_WIN95\_CLASSES 플래그도 동시에 지정해 있습니다만 이번 예에서는 이 두 개의 플래그는 없어도 괜찮습니다.

**▣ 공용 컨트룰러 > 일시 지정**

WTL은 공용 컨트롤의 일시 지정 컨트롤을 CDateTimePickerCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CDateTimePickerCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CDateTimePickerCtrlT<ATL::CWindow> CDateTimePickerCtrl; |

이것은 CDateTimePickerCtrl 클래스는 CWindow클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CDateTimePickerCtrl클래스는 MFC의 CDateTimeCtrl 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CDateTimePickerCtrl클래스를 사용하는 예 입니다. 일시 지정 컨트롤을 드롭 다운하면 디폴트로 월간 캘린더 컨트롤이 표시됩니다만 , 그 하부에 있는 오늘의 일자를 표시하는 부분을 삭제해 표시합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CDateTimePickerCtrl m\_datetime\_getdate;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_HANDLER\_EX(IDC\_DATETIMEPICKER\_GETDATE, DTN\_DROPDOWN, OnDateTimeDropDown)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_datetime\_getdate = GetDlgItem(IDC\_DATETIMEPICKER\_GETDATE);  // 일시 형식의 설정  m\_datetime\_getdate.SetFormat(\_T("yyyy-MM-dd"));  // 초기 상태의 일시 설정  SYSTEMTIME st = {0};  st.wYear = 2004;  st.wMonth = 2;  st.wDay = 7;  m\_datetime\_getdate.SetSystemTime(GDT\_VALID, &st);  return TRUE;  }  LRESULT OnDateTimeDropDown(LPNMHDR pnmh){  CMonthCalendarCtrl calendar = m\_datetime\_getdate.GetMonthCal();  calendar.ModifyStyle(0, MCS\_NOTODAY);  CRect rcCalendar;  calendar.GetMinReqRect(rcCalendar);  calendar.MoveWindow(rcCalendar);  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES | ICC\_DATE\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 일시 지정 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤 명 | 리소스ID |
| 일시 지정 | IDC\_DATETIMEPICKER\_GETDATE |

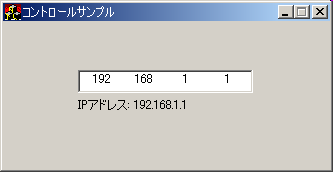
　다음에 stdafx.h 에서는 CDateTimePickerCtrl 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   
  
　CMainDlg클래스에서는 우선 일시 지정 컨트롤용으로 CDateTimePickerCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러로 일시 지정 컨트롤의 일시 표시 포맷을 「yyy-MM-dd」으로 설정합니다. 그리고 SetSystemTime()에 의해 초기 상태의 일시를 2004-02-07로 설정합니다.   
  
　다음에 통지 코드가DTN\_DROPDOWN의 일시 지정 컨트롤용 통지 메세지 핸들러를 추가합니다. 그 핸들러에서는 일시 지정 컨트롤 오브젝트로부터 GetMonthCal()에 의해 월간 캘린더 컨트롤 오브젝트를 취득해 MCS\_NOTODAY스타일을 추가하고 있습니다. 이것에 의해 월간 캘린더 컨트롤의 하부에 있는 오늘의 일자를 표시하는 부분이 공백이 됩니다. 이번 예에서는GetMinReqRect()에 의해 그 공백 부분을 생략한 월간 캘린더 컨트롤 구형을 취득해 MoveWindow()에 의해 그 구형 사이즈로 변경하고 있습니다.   
  
덧붙여 월간 캘린더 컨트롤 오브젝트는 일시 지정 컨트롤의 화살표 버튼을 클릭해 표시(드롭 다운)했을 때 밖에 취득할 수 없습니다.   
  
　마지막으로 \_tWinMain()에서ICC\_DATE\_CLASSES플래그를 지정해 AtlInitCommonControls()를 호출합니다. 이것에 의해 일시 지정 컨트롤이 사용 가능하게 됩니다. 덧붙여 비교적 사용 빈도가 높다고 생각되는ICC\_COOL\_CLASSES플래그와ICC\_WIN95\_CLASSES플래그도 동시에 지정해 있습니다만 이번 예 에서는 이 두 개의 플래그는 없어도 괜찮습니다.

▣ 공용 컨트룰러 > IP어드레스

**< IP 주소 >**  
  
WTL은 공용 컨트롤의 IP주소 컨트롤을 CIPAddressCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CIPAddressCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CIPAddressCtrlT<ATL::CWindow> CIPAddressCtrl; |

이것은 CIPAddressCtrl 클래스는 CWindow클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CIPAddressCtrl클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CIPAddressCtrl클래스를 사용하는 예입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CIPAddressCtrl m\_ip\_show;  CStatic m\_static\_show; // IP주소 표시용  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  NOTIFY\_HANDLER\_EX(IDC\_IPADDRESS\_SHOW, IPN\_FIELDCHANGED, OnIPFieldChanged)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_ip\_show = GetDlgItem(IDC\_IPADDRESS\_SHOW);  m\_static\_show = GetDlgItem(IDC\_STATIC\_SHOW);  m\_ip\_show.SetAddress(MAKEIPADDRESS(192, 168, 1, 1));  return TRUE;  }  LRESULT OnIPFieldChanged(LPNMHDR pnmh){  DWORD dwIP;  m\_ip\_show.GetAddress(&dwIP);  CString strMsg;  strMsg.Format(\_T("IP주소: %d.%d.%d.%d"),  FIRST\_IPADDRESS(dwIP),  SECOND\_IPADDRESS(dwIP),  THIRD\_IPADDRESS(dwIP),  FOURTH\_IPADDRESS(dwIP));  m\_static\_show.SetWindowText(strMsg);  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES | ICC\_INTERNET\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 IP주소 컨트롤과 스태틱 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤 명 | 리소스 ID |
| IP주소 | IDC\_IPADDRESS\_SHOW |
| 스태틱(IP주소 표시용) | IDC\_STATIC\_SHOW |

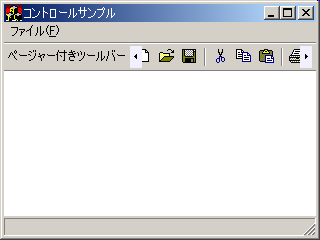
　다음에 stdafx.h 에서는 CIPAddressCtrl 클래스와 CStatic 클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   
  
　CMainDlg 클래스에서는 우선 IP 주소 컨트롤용과 스태틱 컨트롤용으로 각각 CIPAddressCtrl 클래스와 CStatic 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것들을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다.   
  
　다음으로 WM\_INITDIALOG 메세지 핸들러로 IP주소 컨트롤의 초기 상태를 SetAddress()에 의해 192.168.1.1로 설정합니다. 덧붙여 4개의 필드(BYTE값)로부터 SetAddress()에 건네주기 위한 DWORD 값을 얻기 위해서는 MAKEIPADDRESS 매크로를 사용합니다.   
  
　다음으로 통지 코드가 IPN\_FIELDCHANGED의 일시 지정 컨트롤용 통지 메세지 핸들러를 추가합니다. 이 핸들러는 IP 주소 컨트롤의 필드가 변경되었을 때에 불려 갑니다만 , 여기에서는 입력된 IP주소를 GetAddress()에 의해 취득해 매크로로 각 필드 치를 취득하고 나서 스태틱 컨트롤에 표시하고 있습니다.   
  
　마지막으로 \_tWinMain() 에서 ICC\_INTERNET\_CLASSES 플래그를 지정해 AtlInitCommonControls()를 호출합니다. 이것에 의해 IP주소 컨트롤이 사용 가능하게 됩니다. 덧붙여 비교적 사용 빈도가 높다고 생각되는 ICC\_COOL\_CLASSES 플래그와 ICC\_WIN95\_CLASSES 플래그도 동시에 지정해 있습니다만 이번 예 에서는 이 두 개의 플래그는 없어도 괜찮습니다.

**▣ 공용 컨트룰러 > Pager**

WTL은 공용 컨트롤의 pager 컨트롤을 CPagerCtrlT 이라고 하는 템플릿 클래스에서 캡슐화 하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CPagerCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CPagerCtrlT<ATL::CWindow> CPagerCtrl; |

이것은 CPagerCtrl 클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   
  
　이하에 나타내는 것은 CPagerCtrl클래스를 사용하는 예 입니다. 프레임 윈도우를 작성해 그 툴바에 pager를 사용합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // 컨트롤용 클래스를 사용하기 위해  #include <atlframe.h> |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  enum { IDC\_PAGER\_BAR = 1001 };  CPagerCtrl m\_pager\_bar;  CSize m\_sizeToolbar; // 툴바의 사이즈  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateToolBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  NOTIFY\_HANDLER\_EX(IDC\_PAGER\_BAR, PGN\_CALCSIZE, OnPagerCalcsize)  NOTIFY\_HANDLER\_EX(IDC\_PAGER\_BAR, PGN\_SCROLL, OnPagerScroll)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 리버를 작성  CreateSimpleReBar();  // pager 컨트롤을 작성  m\_pager\_bar.Create(m\_hWndToolBar, NULL, NULL,  WS\_CHILD | WS\_VISIBLE | PGS\_HORZ, 0, IDC\_PAGER\_BAR);  // 툴바를 작성  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_pager\_bar.m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  UIAddToolBar(hWndToolBar);  // 툴바의 사이즈를 취득  ::SendMessage(hWndToolBar, TB\_GETMAXSIZE, 0, (LPARAM)&m\_sizeToolbar);  m\_pager\_bar.MoveWindow(0, 0, m\_sizeToolbar.cx, m\_sizeToolbar.cy);  m\_pager\_bar.SetChild(hWndToolBar);  m\_pager\_bar.SetBkColor(RGB(240, 240, 255));  AddSimpleReBarBand(m\_pager\_bar.m\_hWnd, \_T("pager 첨부 툴바"));  // 상태 바를 작성  CreateSimpleStatusBar();  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  LRESULT OnPagerCalcsize(LPNMHDR pnmh){  LPNMPGCALCSIZE pCalcSize = (LPNMPGCALCSIZE)pnmh;  switch(pCalcSize->dwFlag){  case PGF\_CALCWIDTH:  pCalcSize->iWidth = m\_sizeToolbar.cx;  break;  }  return 0;  }  LRESULT OnPagerScroll(LPNMHDR pnmh){  LPNMPGSCROLL pScroll = (LPNMPGSCROLL)pnmh;  switch(pScroll->iDir){  case PGF\_SCROLLLEFT:  case PGF\_SCROLLRIGHT:  case PGF\_SCROLLUP:  case PGF\_SCROLLDOWN:  pScroll->iScroll = 10;  break;  }  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_PAGESCROLLER\_CLASS);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

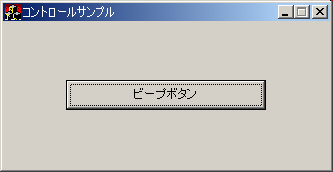
　우선 stdafx.h 에서는 CPagerCtrl 클래스와 프레임 윈도우를 사용하기 위해서 atlctrls.h 헤더와 atlframe.h 헤더를 포함 합니다.   
  
　CMainFrame 클래스에서는 일반적인 프레임 윈도우를 작성하는 것에 더해 pager 컨트롤용으로 ID(IDC\_PAGER\_BAR) 를 정의해 CPagerCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_CREATE 메세지 핸들러로 Create()를 호출해 작성할 필요가 있습니다. 그리고 툴바의 사이즈를 나타내는 CSize 클래스의 인스턴스도 멤버 변수로서 선언해 둡니다.   
  
　다음에 WM\_CREATE 메세지 핸들러로 우선 CreateSimpleReBar()을 호출해 리버를 작성합니다. 다음에 리버를 부모로서 즉 Create()의 제1인수에 리버의 핸들m\_hWndToolBar을 지정해 pager 컨트롤을 작성합니다. 게다가 pager 컨트롤을 부모로서 툴바를 작성합니다. 이 때 스타일로서 CCS\_NORESIZE를 지정할 필요가 있습니다. (ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE 에는 CCS\_NORESIZE 스타일이 포함되어 있습니다.) 다음으로 pager 컨트롤의 사이즈를 툴바와 같은 사이즈로 해 SetChild()에 툴바의 핸들을 지정하는 것에 따라 툴바를 pager 컨트롤의 자식으로 설정합니다. 그리고 SetBkColor에 의해 pager 컨트롤의 색을 지정해 있습니다. 이렇게 작성한 pager 컨트롤을AddSimpleReBarBand에 의해 리버에 추가합니다.   
  
　다음에 통지 코드가 PGN\_CALCSIZE의 pager 컨트롤용 통지 메세지 핸들러를 추가합니다. 이 핸들러에서는 pager 컨트롤의 사이즈를 툴바의 폭과 같은 사이즈로 설정합니다. 이 핸들러는 필수입니다.   
  
　다음에 통지 코드가 PGN\_SCROLL의 pager 컨트롤용 통지 메세지 핸들러를 추가합니다. 이 핸들러에서는 pager 컨트롤의 스크롤치를 설정합니다.   
  
　마지막으로 \_tWinMain()에서 ICC\_PAGESCROLLER\_CLASS 플래그를 지정해 AtlInitCommonControls()를 호출합니다. 이것에 의해 pager 컨트롤이 사용 가능하게 됩니다.

**▣ 컨트룰러 확장 > 슈퍼 클래스 화**

슈퍼 클래스화를 하면 컨트롤 등의 정의가 끝난 윈도우 클래스("BUTTON"나"SysListView32"등 )를 확장하는 새로운 윈도우 클래스를 정의할 수 있습니다.   
  
　이하에 나타내는 것은 버튼 컨트롤을 슈퍼 클래스화 해 클릭되었을 때에 beep음을 울리는 버튼(beep 버튼) 을 정의하는 예입니다. 이 예에서는BeepButton.h이라고 하는 헤더 파일을 준비해 거기에 CBeepButton라고 하는 클래스를 정의하고 있습니다.

|  |
| --- |
| // BeepButton.h  class CBeepButton : public CWindowImpl<CBeepButton>  {  public:  DECLARE\_WND\_SUPERCLASS(\_T("BeepButton"), \_T("BUTTON"))  BEGIN\_MSG\_MAP\_EX(CBeepButton)  MSG\_WM\_LBUTTONDOWN(OnLButtonDown)  END\_MSG\_MAP()  void OnLButtonDown(UINT uFlags, CPoint pt){  MessageBeep(MB\_ICONASTERISK);  SetMsgHandled(FALSE);  }  }; |

슈퍼 클래스화하기 위해서는 DECLARE\_WND\_SUPERCLASS() 매크로를 사용합니다. 이 매크로의 제1인수에는 슈퍼 클래스화하는 윈도우 클래스 명을 지정해 제2인수에는 그 원이 되는 기존의 윈도우 클래스 명을 지정합니다. 이 예 에서는 beep 버튼이 "BUTTON" 윈도우 클래스에 근거하는 것을 의미합니다.   
  
CBeepButton클래스에서는 WM\_LBUTTONDOWN 메세지 핸들러로 beep음을 울리도록 하고 있습니다. 그리고 SetMsgHandled(FALSE)를 호출하는 것에 의해 beep음을 울리는 이외는 보통 버튼과 같이 동작하도록 하고 있습니다.   
  
　다음에 나타내는 것은 이 CBeepButton클래스를 사용하는 예입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h> |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  enum { IDC\_BUTTON\_BEEP = 1001 };  CBeepButton m\_button\_beep;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 작성  m\_button\_beep.Create(m\_hWnd, CRect(0, 0, 200, 30),  \_T("beep 버튼"), WS\_CHILD | WS\_VISIBLE, 0, IDC\_BUTTON\_BEEP);  m\_button\_beep.SetFont(GetFont());  m\_button\_beep.CenterWindow();  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "BeepButton.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 CMainDlg 클래스에서 beep 버튼용으로 CBeepButton 클래스의 인스턴스를 멤버 변수로서 선언합니다. 그리고 그 버튼용의 ID 으로서 IDC\_BUTTON\_BEEP를 선언합니다.   
  
　다음으로 WM\_INITDIALOG 메세지 핸들러로 beep 버튼을 작성합니다.   
  
　마지막으로 Control.cpp파일내에서 maindlg.h 헤더의 앞에 BeepButton.h 헤더를 포함 합니다.   
  
　덧붙여 WTL은 DECLARE\_FRAME\_WND\_SUPERCLASS 이라고 하는 매크로도 준비해 있습니다. 이것은 프레임 윈도우를 슈퍼 클래스화합니다.

**▣ 컨트룰러 확장 > 서브 클래스 화**

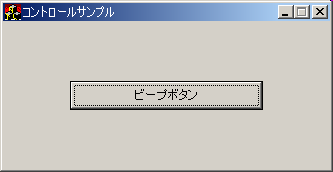
서브 클래스화를 하면 기존의 윈도우(예를 들면 다이얼로그상의 컨트롤) 의 동작을 확장할 수가 있습니다.   
이하에 나타내는 것은 다이얼로그 상의 버튼 컨트롤을 서브 클래스화 해 클릭 되었을 때에 beep음을 울리는 버튼(beep 버튼) 을 작성하는 예 입니다. 이 예 에서는BeepButton.h이라고 하는 헤더 파일을 준비해 거기에 CBeepButton라고 하는 클래스를 정의하고 있습니다.

|  |
| --- |
| // BeepButton.h  class CBeepButton : public CWindowImpl<CBeepButton>  {  public:  BEGIN\_MSG\_MAP\_EX(CBeepButton)  MSG\_WM\_LBUTTONDOWN(OnLButtonDown)  END\_MSG\_MAP()  void OnLButtonDown(UINT uFlags, CPoint pt){  MessageBeep(MB\_ICONASTERISK);  SetMsgHandled(FALSE);  }  }; |

CBeepButton 클래스에서는 WM\_LBUTTONDOWN 메세지 핸들러로 beep음을 울리도록 하고 있습니다. 그리고 SetMsgHandled(FALSE)를 호출하는 것에 의해 beep 음을 울리는 이외는 통상의 동작을 하도록 하고 있습니다.   
  
덧붙여 이 CBeepButton클래스 자체는 통상의 윈도우이며 버튼 컨트롤로 한정된 것이 아닙니다. 실제로 버튼 컨트롤에 근거한 beep 버튼을 작성하기 위해서는 CBeepButton::SubclassWindow()에 버튼 컨트롤의 핸들을 지정해 호출합니다.

|  |
| --- |
| CBeepButton m\_button\_beep;  ...  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  ...  ...  m\_button\_beep.SubclassWindow(기존의 버튼 컨트롤의 핸들);  } |

이와 같이 슈퍼 클래스 화에서는DECLARE\_WND\_SUPERCLASS 매크로를 사용해 「클래스」에 대해서 기본으로 되는 컨트롤을 지정합니다만 서브 클래스화에서는 SubclassWindow()을 사용해 「인스턴스」에 대해서 기본으로 되는 컨트롤을 지정합니다.   
  
　다음에 나타내는 것은 이 CBeepButton클래스를 사용하는 예입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h> |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CBeepButton m\_button\_beep;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 서브 클래스화  m\_button\_beep.SubclassWindow(GetDlgItem(IDC\_BUTTON\_BEEP));  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "BeepButton.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 , 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 푸쉬 버튼 | IDC\_BUTTON\_BEEP |

　다음에 CMainDlg 클래스에서 beep 버튼용으로 CBeepButton 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것은 WM\_INITDIALOG 메세지 핸들러로SubclassWindow()에 의해 서브 클래스화 합니다.   
  
　마지막으로 Control.cpp 파일내에서 maindlg.h 헤더의 앞에 BeepButton.h 헤더를 포함 합니다.

**▣ 컨트룰러 확장 > 컨테인 윈도우 < ContainedWindow >**

ContainedWindow는 받은 모든 메세지를 자신의 컨테이너 윈도우에 건네줍니다. 일반적으로 컨테이너 윈도우는 부모 윈도우 이므로 ContainedWindow는 그 자식 윈도우 입니다만 그렇지 않은 경우도 있습니다. 컨테이너 윈도우의 클래스는 ContainedWindow의 클래스를 멤버 변수로서 가집니다.   
  
ContainedWindow는 컨트롤 등의 정의가 끝난 윈도우 클래스 ("BUTTON"나"SysListView32"등 ) 를 기본으로 합니다. 다음에 나타내는 것은 다이얼로그를 컨테이너로 했을 때의 예 입니다.

|  |
| --- |
| // 컨테이너 윈도우  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CContainedWindow m\_contained; // 콘테 인도 윈도우  // constructor  CMainDlg() : m\_contained(\_T("BUTTON"), this, 100)  {}  ...  ... |

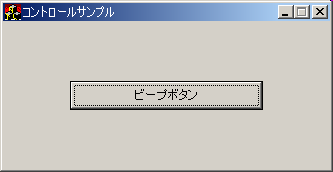
　우선 , ContainedWindow의 인스턴스를 컨테이너 윈도우(이 예에서는 다이얼로그) 클래스의 멤버 변수로서 선언합니다. 그리고 컨테이너 윈도우의 constructor　 　 으로 초기화 합니다. 제1인수에는 정의가 끝난 윈도우 클래스 명을 제2인수에는 메세지를 처리하는 윈도우에의 포인터를 제3인수에는 메세지를 처리하는 대체 메세지 맵의 번호를 지정 합니다.   
  
이 예 에서는 메세지를 처리하는 대체 메세지 맵의 번호를 100로 지정하고 있습니다. 따라서 CMainDlg 클래스의 메세지 맵은 다음과 같이 합니다.

|  |
| --- |
| Class CMainDlg : public CDialogImpl<CMainDlg>  {  ...  ...  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  ...  ...  ALT\_MSG\_MAP(100) // ContainedWindow용 대체 메세지 맵  MSG\_WM\_LBUTTONDOWN(OnLButtonDownForContained)  END\_MSG\_MAP()  void OnLButtonDownForContained(UINT uFlags, CPoint pt){  MessageBeep(MB\_ICONASTERISK);  SetMsgHandled(FALSE);  }  ...  ... |

이것에 의해 ContainedWindow의 메세지는 번호100의 대체 메세지 맵으로 처리됩니다. 즉 , ContainedWindow (이 예에서는"BUTTON"을 지정했으므로 버튼 컨트롤) 를 왼쪽 클릭하면 OnLButtonDownForContained()가 불려 갑니다.   
  
마지막으로 ContainedWindow를 작성할 필요가 있습니다. 작성하기 위해서는 Create()을 호출합니다만 SubclassWindow()에 의해 기존의 컨트롤을 서브 클래스화할 수도 있습니다.

|  |
| --- |
| class CMainDlg : public CDialogImpl<CMainDlg>  {  ...  ...  enum { IDC\_BUTTON\_BEEP = 1001 };  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 컨트롤 작성  m\_contained.Create(m\_hWnd, CRect(0, 0, 200, 30),  \_T("beep 버튼"), WS\_CHILD | WS\_VISIBLE, 0, IDC\_BUTTON\_BEEP);  // 또는 기존의 컨트롤을 서브 클래스화  // m\_contained.SubclassWindow(GetDlgItem(IDC\_BUTTON\_BEEP));  return TRUE;  }  ...  ... |

이와 같이 ContainedWindow를 사용하면 CBeepButton와 같이 새로운 클래스를 만들지 않아도 컨트롤의 동작을 변경할 수가 있습니다.   
  
　다음에 나타내는 것은 클릭되었을 때에 beep 음을 울리는 버튼(beep 버튼)을 ContainedWindow를 사용해 작성하는 예입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h> |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CContainedWindow m\_contained;  // constructor  CMainDlg() : m\_contained(\_T("BUTTON"), this, 100)  {}  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  ALT\_MSG\_MAP(100)  MSG\_WM\_LBUTTONDOWN(OnLButtonDownForContained)  END\_MSG\_MAP()  void OnLButtonDownForContained(UINT uFlags, CPoint pt){  MessageBeep(MB\_ICONASTERISK);  SetMsgHandled(FALSE);  }  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 서브 클래스화  m\_contained.SubclassWindow(GetDlgItem(IDC\_BUTTON\_BEEP));  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp내  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 푸쉬 버튼 | IDC\_BUTTON\_BEEP |

　다음에 CMainDlg 클래스에서 beep 버튼 용으로 ContainedWindow 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것은 CMainDlg 클래스의 constructor　 　 으로 초기화합니다.   
  
　다음에 ContainedWindow 용의 대체 메세지 맵과 메세지 핸들러를 준비해 마지막에 WM\_INITDIALOG 메세지 핸들러로 SubclassWindow()에 의해 기존의 버튼 컨트롤을 서브 클래스 화 합니다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **▣ 컨트룰러 확장 > 메시지 리프렉션 < 메세지 리플렉션 >**  통상 컨트롤은 WM\_COMMAND(코먼 컨트롤의 경우는WM\_NOTIFY) 메세지를 부모 윈도우에 송신합니다. 부모 윈도우는 그러한 메세지에 응답해 처리를 실행합니다.   　그러나 컨트롤이 보내는 그러한 메세지를 부모 윈도우는 아니고 그 컨트롤 자신에게 응답시키고 싶은 경우가 있습니다. 메세지 리플렉션은 이것을 실현하는 구조입니다. 메세지 리플렉션을 사용하면 컨트롤이 부모 윈도우에 보낸 메세지는 부모 윈도우로부터 컨트롤에 반송되어 컨트롤이 응답할 수가 있습니다.   다음에 나타내는 것은 컨트롤로부터의 메세지를 반송하는 부모 윈도우 클래스의 메세지 맵입니다.   |  | | --- | | // 부모 윈도우  class CMainDlg : public CDialogImpl<CMainDlg>  {  ...  ...  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  ...  ...  REFLECT\_NOTIFICATIONS() // 처리되지 않았던 컨트롤로부터의 메세지를 반송  END\_MSG\_MAP()  ... ... |   　부모 윈도우(이 예에서는CMainDlg)는 컨트롤로부터의 메세지를 받아들이면 메세지 맵에 그 메시지 용의 메세지 맵 엔트리가 있는지 어떤지 조사합니다. 해당하는 메세지 맵 엔트리가 없으면 , REFLECT\_NOTIFICATIONS 매크로에 의해 메세지를 보내 온 컨트롤에 반송합니다.   　다음에 반송된 메세지에 응답하는 컨트롤 클래스를 정의합니다. 여기에서는 버튼용 클래스CUrlButton를 정의합니다. CUrlButton클래스는 버튼의 캡션 캐릭터 라인을 URL이라고 가정해 버튼을 누르면 그 URL를 엽니다. 이 예에서는 UrlButton.h이라고 하는 헤더 파일을 준비해 거기에 CUrlButton클래스를 정의하고 있습니다.   |  | | --- | | // UrlButton.h  class CUrlButton : public CWindowImpl<CUrlButton>  {  public:  DECLARE\_WND\_SUPERCLASS(\_T("UrlButton"), \_T("BUTTON"))  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CUrlButton)  MSG\_OCM\_COMMAND(OnCommand)  DEFAULT\_REFLECTION\_HANDLER()  END\_MSG\_MAP()  void OnCommand(UINT uNotifyCode, int nID, HWND hWndCtl){  TCHAR szURL[2048];  GetWindowText(szURL, sizeof(szURL)/sizeof(TCHAR));  ShellExecute(m\_hWnd, \_T("open"), szURL, NULL, NULL, SW\_SHOWNORMAL);  }  } |   이것에 의해 부모 윈도우로부터 반송된 메세지는 메세지를 보낸 컨트롤의 메세지 맵으로 처리됩니다. 반송된 메시지는 OCM\_으로 시작됩니다. (OCM\_COMMAND나OCM\_NOTIFY나OCM\_CTLCOLORBTN등. ) WTL는 반송된 메세지를 위한 메세지 매크로를 준비해 있습니다. 위의 예에서는OCM\_COMMAND 메시지를 MSG\_OCM\_COMMAND 메세지 매크로에 의해 OnCommand()에 매핑 하고 있습니다. 덧붙여 처리되지 않았던 메시지는 DEFAULT\_REFLECTION\_HANDLER 매크로에 의해 처리됩니다.   마지막으로 컨트롤을 작성할 필요가 있습니다. 작성하기 위해서는Create()을 호출합니다만 SubclassWindow()에 의해 기존의 컨트롤을 서브 클래스화할 수도 있습니다.   |  | | --- | | class CMainDlg : public CDialogImpl<CMainDlg>  {  ...  ...  enum { IDC\_BUTTON\_URL = 1001 };  CUrlButton m\_button\_url;  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 컨트롤 작성  m\_button\_url.Create(m\_hWnd, CRect(0, 0, 200, 30),  \_T("http://www.google.com/"), WS\_CHILD | WS\_VISIBLE, 0, IDC\_BUTTON\_URL);  // 또는 기존의 컨트롤을 서브 클래스화  // m\_button\_url.SubclassWindow(GetDlgItem(IDC\_BUTTON\_URL));  return TRUE;  }  ... ... |   이와 같이 메세지 리플렉션을 사용하면 컨트롤 클래스가 부모 윈도우에 의존하지 않고 자신이 부모 윈도우에 보낸 메세지에 대한 동작을 자기 자신으로 구현 할 수 있습니다.   　다음에 나타내는 것은 CUrlButton클래스를 사용하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl11-04/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h> |  |  | | --- | | // maindlg.h내  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CUrlButton m\_button\_url;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  REFLECT\_NOTIFICATIONS() // 메세지를 반송  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 서브 클래스화  m\_button\_url.SubclassWindow(GetDlgItem(IDC\_BUTTON\_URL));  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "UrlButton.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치하고 리소수 ID를 다음과 같이 지정합니다. 덧붙여 버튼의 캡션은 유효한URL캐릭터 라인으로 설정합니다.   |  |  | | --- | --- | | 컨트롤명 | 리소스 ID | | 푸쉬 버튼 | IDC\_BUTTON\_URL |   　다음에 CMainDlg 클래스에서 URL버튼용으로 CUrlButton 클래스의 인스턴스를 멤버 변수로서 선언합니다.   　다음에 메세지 맵으로 REFLECT\_NOTIFICATIONS를 추가해 마지막에 WM\_INITDIALOG 메세지 핸들러로 SubclassWindow()에 의해 기존의 버튼 컨트롤을 서브 클래스화합니다.   　그런데 이번은 반송된 메세지의 매핑에 MSG\_OCM\_COMMAND 메세지 매크로를 사용했습니다만 WTL은 반송된 메시지용 으로 다음과 같은 종류의 메세지 매크로를 준비해 있습니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 메세지 리플렉션용 커멘드 핸들러 매크로 |
| 다음의 메세지 매크로는 , 반송된 커멘드 메세지(OCM\_COMMAND) 을 대상으로 합니다.  MSG\_OCM\_COMMAND(핸들러명)  반송된 커멘드 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_COMMAND\_HANDLER\_EX(ID, 통지 코드, 핸들러명)  지정된 컨트롤로부터의 지정된 통지 코드를 가지는 반송된 커멘드 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_COMMAND\_ID\_HANDLER\_EX(ID, 핸들러명)  지정된 컨트롤로부터의 임의의 통지 코드를 가지는 반송된 커멘드 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_COMMAND\_CODE\_HANDLER\_EX(통지 코드, 핸들러명)  임의의 컨트롤로부터의 지정된 통지 코드를 가지는 반송된 커멘드 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_COMMAND\_RANGE\_HANDLER\_EX(개시 위치의 메세지명, 종료 위치의 메세지명, 핸들러명)  연속한 범위의 컨트롤로부터의 반송된 커멘드 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_COMMAND\_RANGE\_CODE\_HANDLER\_EX(개시 위치의 메세지명, 종료 위치의 메세지명, 통지 코드, 핸들러명)  연속한 범위의 컨트롤로부터의 지정된 통지 코드를 가지는 반송된 커멘드 메세지를 핸들러 함수에 맵 합니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 메세지 리플렉션용 통지 핸들러 매크로 |
| 다음의 메세지 매크로는 , 반송된 통지 메세지(OCM\_NOTIFY) 를 대상으로 합니다.  MSG\_OCM\_NOTIFY(핸들러명)  반송된 통지 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_NOTIFY\_HANDLER\_EX(ID, 통지 코드, 핸들러명)  지정된 컨트롤로부터의 지정된 통지 코드를 가지는 반송된 통지 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_NOTIFY\_ID\_HANDLER\_EX(ID, 핸들러명)  지정된 컨트롤로부터의 임의의 통지 코드를 가지는 반송된 통지 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_NOTIFY\_CODE\_HANDLER\_EX(통지 코드, 핸들러명)  임의의 컨트롤로부터의 지정된 통지 코드를 가지는 반송된 통지 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_NOTIFY\_RANGE\_HANDLER\_EX(개시 위치의 메세지명, 종료 위치의 메세지명, 핸들러명)  연속한 범위의 컨트롤로부터의 반송된 통지 메세지를 핸들러 함수에 맵 합니다.  REFLECTED\_NOTIFY\_RANGE\_CODE\_HANDLER\_EX(개시 위치의 메세지명, 종료 위치의 메세지명, 통지 코드, 핸들러명)  연속한 범위의 컨트롤로부터의 지정된 통지 코드를 가지는 반송된 통지 메세지를 핸들러 함수에 맵 합니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 그 외의 메세지 리플렉션용 핸들러 매크로 |
| 다음의 메세지 매크로는 , 앞에서 말한 것 이외의 메세지 리플렉션용 메세지를 대상으로 합니다.   * MSG\_OCM\_PARENTNOTIFY(핸들러명) * MSG\_OCM\_DRAWITEM(핸들러명) * MSG\_OCM\_MEASUREITEM(핸들러명) * MSG\_OCM\_COMPAREITEM(핸들러명) * MSG\_OCM\_DELETEITEM(핸들러명) * MSG\_OCM\_VKEYTOITEM(핸들러명) * MSG\_OCM\_CHARTOITEM(핸들러명) * MSG\_OCM\_HSCROLL(핸들러명) * MSG\_OCM\_VSCROLL(핸들러명) * MSG\_OCM\_CTLCOLOREDIT(핸들러명) * MSG\_OCM\_CTLCOLORLISTBOX(핸들러명) * MSG\_OCM\_CTLCOLORBTN(핸들러명) * MSG\_OCM\_CTLCOLORDLG(핸들러명) * MSG\_OCM\_CTLCOLORSCROLLBAR(핸들러명) * MSG\_OCM\_CTLCOLORSTATIC(핸들러명) | |

**▣ 컨트룰러 확장 > DDX / DDV**MFC에서는 DDC/DDV라고 하는 구조가 준비되어 있습니다만, WTL에서도 DDX/DDV가 준비되어 있습니다.  
다만 MFC의 그것과는 조금 사용 방법이 다릅니다. 여기에서는 WTL의 DDX/DDV에 대해서 설명 합니다.

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | DDX |
| DDX(dialog data exchange) 는 다이얼로그 상의 컨트롤의 초기화와 컨트롤과의 데이터의 교환을 단순화하기 위한 구조입니다.   여기서 말하는 컨트롤의 초기화란 서브 클래스화하거나 윈도우 핸들을 컨트롤 클래스로 설정하거나 하는 것을 의미합니다. 컨트롤과의 데이터의 교환이란 지정한 변수의 값을 컨트롤로 설정하거나 컨트롤로부터 취득한 값을 지정한 변수에 대입하는 것을 의미합니다.   　다음에 나타내는 것은 이DDX를 사용하는 예입니다. 1로부터 에디트 컨트롤에 입력한 값까지의 총화를 요구합니다. (예를 들면10을 입력했을 경우 ,1+2+3+···+10(을)를 계산. ) 덧붙여 값의 상한은100으로 합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl11-05/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlddx.h> // DDX/DDV을 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>, public CWinDataExchange<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // constructor  CMainDlg() : m\_nInput(1)  {}  CEdit m\_edit\_input;  int m\_nInput;  // DDX맵  BEGIN\_DDX\_MAP(CMainDlg)  DDX\_CONTROL\_HANDLE(IDC\_EDIT\_INPUT, m\_edit\_input)  DDX\_INT(IDC\_EDIT\_INPUT, m\_nInput)  END\_DDX\_MAP()  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_SUM, OnButtonSum)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  DoDataExchange(FALSE);  m\_edit\_input.LimitText(3);  return TRUE;  }  void OnDataExchangeError(UINT nCtrlID, BOOL bSave){  CString strMsg;  strMsg.Format(\_T("컨트롤(ID:%u) 과의 데이터 교환에 실패. "), nCtrlID);  MessageBox(strMsg, \_T("DDX에러"), MB\_ICONWARNING);  ::SetFocus(GetDlgItem(nCtrlID));  }  void OnButtonSum(UINT uNotifyCode, int nID, HWND hWndCtl){  if(DoDataExchange(TRUE)){  // 치 체크  if(m\_nInput < 1 || 100 < m\_nInput){  MessageBox(\_T("1 로부터 100 까지 의 값을 입력해 주세요. "),  \_T("에러"), MB\_ICONWARNING);  return;  }  int nSum = 0;  for(int i=1; i<=m\_nInput; i++)  nSum += i;  CString strMsg;  strMsg.Format(\_T("1 로부터 %d 까지 의 총화는 %d 입니다. "), m\_nInput, nSum);  MessageBox(strMsg);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 리소스를 작성합니다. 다이얼로그에 에디트 컨트롤과 버튼 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 에디트 컨트롤의 [스타일]에서는 디폴트에 더해 [번호]에 체크를 넣습니다.   |  |  | | --- | --- | | 컨트롤명 | 리소스 ID | | 에디트 | IDC\_EDIT\_INPUT | | 푸쉬 버튼 | IDC\_BUTTON\_SUM |   　다음으로 DDX/DDV를 사용하기 위해서 stdafx.h안에서 atlddx.h헤더를 포함 합니다.   　다음으로 CMainDlg클래스의 base class에 CWinDataExchange 클래스를 추가합니다. 그리고 CEdit 클래스의 인스턴스와 int 형태의 변수를 CMainDlg 클래스의 멤버 변수로서 선언합니다. int형태 변수는 CMainDlg 클래스의 constructor　 　 으로 1 에 초기화합니다.   CEdit 클래스의 인스턴스는 DDX 맵의 DDX\_CONTROL\_HANDLE매크로에 의해 초기화됩니다. DDX\_CONTROL\_HANDLE 매크로는 제1인수에게 건네졌다 ID의 컨트롤의 윈도우 핸들을 제2인수에게 건네진 오브젝트에 대입합니다.   WTL의DDX는 컨트롤의 초기화용으로 다음과 같은 매크로를 준비해 있습니다.   * DDX\_CONTROL(ID, 오브젝트명)   지정된 ID의 컨트롤을 지정된 오브젝트에 서브 클래스화합니다.   * DDX\_CONTROL\_HANDLE(ID, 오브젝트명)   지정된 ID의 컨트롤의 윈도우 핸들을 지정된 오브젝트에 대입합니다.  덧붙여 이러한 매크로가 실행되는 (즉 컨트롤 클래스의 인스턴스가 초기화된다 ) 것은 컨트롤 클래스의 인스턴스에 NULL가 설정되어 있어 , 한편 DoDataExchange(FALSE)을 호출한 때만입니다. 위의 예에서는 OnInitDialog() 안에서DoDataExchange(FALSE)를 호출해 CEdit 클래스의 인스턴스인m\_edit\_input에 에디트 컨트롤의 윈도우 핸들을 설정해 있습니다. 그 후CEdit::LimitText()를 호출해 에디트 컨트롤에 3자리수 밖에 입력할 수 없게 하고 있습니다.   　CMainDlg클래스의 DDX맵에서는 또한 DDX\_INT매크로를 사용해 컨트롤과int형태 변수를 관련짓고 있습니다. 이렇게 하는 것에 의해 ,DoDataExchange(FALSE)를 호출했을 때는 자동적으로SetDlgItemInt()에 의해 변수의 값이 컨트롤로 설정되어 DoDataExchange(TRUE)를 호출했을 때는 자동적으로GetDlgItemInt()에 의해 컨트롤에 입력된 수치가 변수에 대입됩니다. 값의 취득 또는 설정에 성공했을 때는 ,DoDataExchange()은TRUE을 돌려줍니다. 실패했을 때는FALSE을 돌려줍니다.   WTL의DDX는 컨트롤과 변수가 관련짓고용으로 다음과 같은 매크로를 준비해 있습니다.   * DDX\_TEXT(ID, 변수명)   지정된 ID의 컨트롤과 지정된 캐릭터 라인형 변수(LPTSTR,BSTR,CComBSTR,CString) 를 관련짓습니다.   * DDX\_INT(ID, 변수명)   지정된 ID의 컨트롤과 지정되었다 int형태 변수를 관련짓습니다.   * DDX\_UINT(ID, 변수명)   지정된 ID의 컨트롤과 지정된UINT형태 변수를 관련짓습니다.   * DDX\_FLOAT(ID, 변수명)   지정된 ID의 컨트롤과 지정된float형태 변수를 관련짓습니다.   * DDX\_FLOAT\_P(ID, 변수명, 정밀도)   지정된 ID의 컨트롤과 지정된float형태 변수를 관련짓습니다. 정밀도를 지정할 수 있습니다.   * DDX\_CHECK(ID, 변수명)   지정된 ID의 체크 버튼 컨트롤과 지정된int형태 또는bool형태 변수를 관련짓습니다.   * DDX\_RADIO(ID, 변수명)   지정된 ID의 라디오 버튼 컨트롤과 지정된int형태 변수를 관련짓습니다.  덧붙여 컨트롤과 변수의 사이에 데이터의 취득 또는 설정이 실패했을 때는 CWinDataExchange 클래스의 멤버 함수인 OnDataExchangeError()이 불려 갑니다. CMainDlg클래스에서는 이OnDataExchangeError()를 오버라이드(override) 해 메시지 박스를 표시하도록(듯이) 하고 있습니다.  마지막으로 버튼의 핸들러 함수인OnButtonSum()을 추가합니다. OnButtonSum()안에서는 입력된 값을 체크해 유효한 값이면 1 으로부터 그 값까지의 총화를 계산해 메시지 박스로 표시합니다. | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | DDV |
| DDV(dialog data validation) (은)는 , 다이얼로그상의 컨트롤에 입력된 데이터의 정당성을 체크하기 위한 구조입니다.   앞의 것DDX의 예에서는 , 에디트 컨트롤에 입력된 수치의 범위 체크를 OnButtonSum()로 다음과 같이 가고 있었습니다.   |  | | --- | | void OnButtonSum(UINT uNotifyCode, int nID, HWND hWndCtl){  if(DoDataExchange(TRUE)){  // 값 체크  if(m\_nInput < 1 || 100 < m\_nInput){  MessageBox(\_T("1 로부터 100 까지 의 값을 입력해 주세요. "),  \_T("에러"), MB\_ICONWARNING);  return;  }  ... ... |   DDV를 사용하면(자) , 이러한 값의 체크 처리를 간략화할 수 있습니다. 다음에 나타내는 것은DDV을 사용해 값의 체크를 하는 예입니다. 앞의 것DDX의 예의CMainDlg클래스만을 변경합니다.   |  | | --- | | // maindlg.h안  class CMainDlg : public CDialogImpl<CMainDlg>, public CWinDataExchange<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // constructor  CMainDlg() : m\_nInput(1)  {}  CEdit m\_edit\_input;  int m\_nInput;  // DDX맵  BEGIN\_DDX\_MAP(CMainDlg)  DDX\_CONTROL\_HANDLE(IDC\_EDIT\_INPUT, m\_edit\_input)  DDX\_INT\_RANGE(IDC\_EDIT\_INPUT, m\_nInput, 1, 100)  END\_DDX\_MAP()  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_SUM, OnButtonSum)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  DoDataExchange(FALSE);  m\_edit\_input.LimitText(3);  return TRUE;  }  void OnDataExchangeError(UINT nCtrlID, BOOL bSave){  CString strMsg;  strMsg.Format(\_T("컨트롤(ID:%u) 과의 데이터 교환에 실패. "), nCtrlID);  MessageBox(strMsg, \_T("DDX에러"), MB\_ICONWARNING);  ::SetFocus(GetDlgItem(nCtrlID));  }  void OnDataValidateError(UINT nCtrlID, BOOL bSave, \_XData& data){  CString strMsg;  strMsg.Format(\_T("%d 로부터 %d 까지 의 값을 입력해 주세요. "),  data.intData.nMin, data.intData.nMax);  MessageBox(strMsg, \_T("DDV에러"), MB\_ICONEXCLAMATION);  ::SetFocus(GetDlgItem(nCtrlID));  }  void OnButtonSum(UINT uNotifyCode, int nID, HWND hWndCtl){  if(DoDataExchange(TRUE)){  int nSum = 0;  for(int i=1; i<=m\_nInput; i++)  nSum += i;  CString strMsg;  strMsg.Format(\_T("1 로부터 %d 까지 의 총화는 %d 입니다. "), m\_nInput, nSum);  MessageBox(strMsg);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |   우선 ,DDX맵의DDX\_INT매크로를 ,DDX\_INT\_RANGE매크로로 변경합니다. 이 매크로의 제1, 2인수는DDX\_INT매크로와 같이 , 컨트롤의 것ID과int형태 변수를 지정합니다. 제3인수에는 값의 최소치를 , 제4인수에는 값의 최대치를 지정합니다.   이렇게 하는 것에 의해 ,DoDataExchange()를 호출해 컨트롤에 값을 설정하거나 컨트롤로부터 값을 취득할 때에 , 그 값의 범위 체크를 합니다. 정당한 값 때는 ,DoDataExchange()은TRUE을 돌려줍니다. 부정한 값 때는FALSE을 돌려줍니다.   WTL(은)는 다음과 같은DDV기능부DDX매크로를 준비해 있습니다.   * DDX\_TEXT\_LEN(ID, 변수명, 문자수)   지정되었다ID의 컨트롤과 지정된 캐릭터 라인형 변수(LPTSTR,BSTR,CComBSTR,CString) 를 관련짓습니다. 문자수 체크를 합니다.   * DDX\_INT\_RANGE(ID, 변수명, 최소치, 최대치)   지정되었다ID의 컨트롤과 지정된int형태 변수를 관련짓습니다. 값의 범위 체크를 합니다.   * DDX\_UINT\_RANGE(ID, 변수명, 최소치, 최대치)   지정되었다ID의 컨트롤과 지정된UINT형태 변수를 관련짓습니다. 값의 범위 체크를 합니다.   * DDX\_FLOAT\_RANGE(ID, 변수명, 최소치, 최대치)   지정되었다ID의 컨트롤과 지정된float형태 변수를 관련짓습니다. 값의 범위 체크를 합니다.   * DDX\_FLOAT\_P\_RANGE(ID, 변수명, 최소치, 최대치, 정밀도)   지정되었다ID의 컨트롤과 지정된float형태 변수를 관련짓습니다. 정밀도를 지정할 수 있습니다. 값의 범위 체크를 합니다.  DDV에 의해 값이 부정이라고 판단되었을 경우는 , CWinDataExchange클래스의 멤버 함수인OnDataValidateError()이 불려 갑니다. CMainDlg클래스에서는 이OnDataValidateError()를 오버라이드(override) 해 , 메시지 박스를 표시하도록(듯이) 하고 있습니다. | |

▣ 컨트룰러 확장 > 오너 드로우

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 오너 드로우를 이용하면 컨트롤의 외관을 독자적인 것으로 할 수가 있습니다. WTL은 오너 드로우를 서포트 하기 위해서 COwnerDraw클래스를 준비해 있습니다.   　다음에 나타내는 것은 COwnerDraw 클래스를 사용해 파랑과 빨강의 푸쉬 버튼을 작성하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl11-06/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // COwnerDraw클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>, public COwnerDraw<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  CHAIN\_MSG\_MAP(COwnerDraw<CMainDlg>) // COwnerDraw클래스에 체인  END\_MSG\_MAP()  void DrawItem(LPDRAWITEMSTRUCT lpdis){  if(lpdis->CtlType == ODT\_BUTTON){  switch(lpdis->CtlID){  case IDC\_BUTTON\_BLUE:  DrawButton(lpdis, RGB(255, 255, 255), RGB(0, 0, 255));  break;  case IDC\_BUTTON\_RED:  DrawButton(lpdis, RGB(255, 255, 255), RGB(255, 0, 0));  break;  default:  break;  }  }  }  void DrawButton(LPDRAWITEMSTRUCT lpdis, COLORREF clrText, COLORREF clrBg){  CDCHandle dc = lpdis->hDC;  dc.SaveDC();    // 캡션 표시 영역을 초기화  CRect rcCaptionArea = lpdis->rcItem;  rcCaptionArea.DeflateRect(2, 2);  // 배경색 묘화  CBrush brushBg;  brushBg.CreateSolidBrush(clrBg);  dc.FillRect(&lpdis->rcItem, brushBg);  // 3D연 묘화  if(lpdis->itemState & ODS\_SELECTED){  rcCaptionArea.OffsetRect(1, 1);  dc.DrawEdge(&lpdis->rcItem, EDGE\_SUNKEN, BF\_RECT);  }else{  dc.DrawEdge(&lpdis->rcItem, EDGE\_RAISED, BF\_RECT);  }  // 포커스 구형 묘화  if(lpdis->itemState & ODS\_FOCUS){  CRect rcFocus = lpdis->rcItem;  rcFocus.DeflateRect(4, 4);  dc.DrawFocusRect(rcFocus);  }  // 캡션 묘화  dc.SetTextColor(clrText);  dc.SetBkMode(TRANSPARENT);  dc.DrawText(\_T("오나드로보탄"), -1,  rcCaptionArea, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  dc.RestoreDC(-1);  }  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 2개 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 양쪽 모두의 버튼 컨트롤의 [스타일]에서는 디폴트에 [오너 묘화]에 체크를 넣습니다.   |  |  | | --- | --- | | 컨트롤명 | 리소스 ID | | 푸쉬 버튼 | IDC\_BUTTON\_BLUE | | 푸쉬 버튼 | IDC\_BUTTON\_RED |   　다음에 COwnerDraw 클래스를 사용하기 위해서 stdafx.h안에서 atlframe.h헤더를 포함 합니다.   　다음에 CMainDlg 클래스의 base class에 COwnerDraw클래스를 추가해 메세지 맵에 COwnerDraw클래스에 체인을 추가합니다. COwnerDraw클래스는 내부에 다음과 같은 메세지 맵을 준비해 있습니다.   |  | | --- | | // COwnerDraw클래스  BEGIN\_MSG\_MAP(COwnerDraw< T >)  MESSAGE\_HANDLER(WM\_DRAWITEM, OnDrawItem)  MESSAGE\_HANDLER(WM\_MEASUREITEM, OnMeasureItem)  MESSAGE\_HANDLER(WM\_COMPAREITEM, OnCompareItem)  MESSAGE\_HANDLER(WM\_DELETEITEM, OnDeleteItem)  ALT\_MSG\_MAP(1) // 메세지 리플렉션용 대체 메세지 맵  MESSAGE\_HANDLER(OCM\_DRAWITEM, OnDrawItem)  MESSAGE\_HANDLER(OCM\_MEASUREITEM, OnMeasureItem)  MESSAGE\_HANDLER(OCM\_COMPAREITEM, OnCompareItem)  MESSAGE\_HANDLER(OCM\_DELETEITEM, OnDeleteItem)  END\_MSG\_MAP() |   4개의 메세지 핸들러는 각각 다음과 같은 멤버 함수를 호출합니다.   |  |  | | --- | --- | | 메세지 핸들러명 | 멤버 함수명 | | OnDrawItem | DrawItem | | OnMeasureItem | MeasureItem | | OnCompareItem | CompareItem | | OnDeleteItem | DeleteItem |   COwnerDraw 클래스에의 체인에 의해 오너 드로우가 필요한 타이밍에 메시지가 COwnerDraw 클래스에 보내져 이러한 멤버 함수가 실행됩니다. COwnerDraw클래스의 상속 클래스(위의 예에서는CMainDlg클래스) 에서는 이러한 멤버 함수를 오버라이드(override) 하는 것에 의해 컨트롤에 독자적인 외관을 줍니다.   CMainDlg클래스에서는 DrawItem()을 오버 라이드(override) 해 컨트롤의 ID별로 독자적인 푸쉬 버튼을 묘화 하고 있습니다.   　그런데 COwnerDraw 클래스 내부의 메세지 맵에는 메세지 리플렉션용의 대체 메세지 맵이 준비되어 있습니다. 다음에 나타내는 것은 메세지 리플렉션을 사용한 오너 드로우의 방법입니다. |

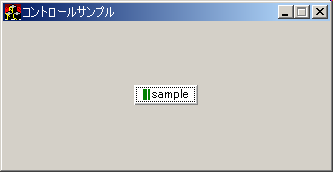
|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 메세지 리플렉션에 의한 오너 드로우 |
| 메세지 리플렉션을 이용하는 것에 의해 컨트롤 자신에게 오너 드로우 시킬 수가 있기 때문에 재사용 가능한 컨트롤 클래스를 기술하기 쉬워집니다. 다음에 나타내는 것은 COwnerDraw 클래스와 메세지 리플렉션을 이용해 색 변경이 가능한 버튼 컨트롤 클래스CColorButton의 정의하는 예 입니다. 이 예에서는 ColorButton.h이라고 하는 헤더 파일을 준비해 거기에 CColorButton 클래스를 정의하고 있습니다.   |  | | --- | | // ColorButton.h  class CColorButton : public CWindowImpl<CColorButton>, public COwnerDraw<CColorButton>  {  public:  DECLARE\_WND\_SUPERCLASS(\_T("ColorButton"), \_T("BUTTON"))  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CColorButton)  CHAIN\_MSG\_MAP\_ALT(COwnerDraw<CColorButton>, 1)  DEFAULT\_REFLECTION\_HANDLER()  END\_MSG\_MAP()  COLORREF m\_clrText;  COLORREF m\_clrBg;  CColorButton() : m\_clrText(RGB(0, 0, 0)), m\_clrBg(RGB(255, 255, 255))  {}  void DrawItem(LPDRAWITEMSTRUCT lpdis){  CDCHandle dc = lpdis->hDC;  dc.SaveDC();    // 캡션 표시 영역을 초기화  CRect rcCaptionArea = lpdis->rcItem;  rcCaptionArea.DeflateRect(2, 2);  // 배경색 묘화  CBrush brushBg;  brushBg.CreateSolidBrush(m\_clrBg);  dc.FillRect(&lpdis->rcItem, brushBg);  // 3D연 묘화  if(lpdis->itemState & ODS\_SELECTED){  rcCaptionArea.OffsetRect(1, 1);  dc.DrawEdge(&lpdis->rcItem, EDGE\_SUNKEN, BF\_RECT);  }else{  dc.DrawEdge(&lpdis->rcItem, EDGE\_RAISED, BF\_RECT);  }  // 포커스 구형 묘화  if(lpdis->itemState & ODS\_FOCUS){  CRect rcFocus = lpdis->rcItem;  rcFocus.DeflateRect(4, 4);  dc.DrawFocusRect(rcFocus);  }  // 캡션 묘화  dc.SetTextColor(m\_clrText);  dc.SetBkMode(TRANSPARENT);  dc.DrawText(\_T("오나드로보탄"), -1,  rcCaptionArea, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  dc.RestoreDC(-1);  }  void SetColor(COLORREF clrText, COLORREF clrBg){  m\_clrText = clrText;  m\_clrBg = clrBg;  }  } |   이 CColorButton클래스는 CWindowImpl 클래스와 COwnerDraw클래스로부터 파생하고 있습니다. CColorButton 클래스는 반송된 오너 드로우용 메세지를 체인에 의해COwnerDraw클래스의 대체 메세지 맵에 보냅니다. 대체 메세지 맵에는 전제와 같이 4개의 메세지 핸들러가 준비되어 있어 CColorButton 클래스는 그 중의 OnDrawItem() 이 호출하는 DrawItem()을 오버라이드(override) 해 버튼을 묘화 하고 있습니다.   　다음에 이 CColorButton 클래스를 사용해 앞의 오너 드로우 버튼의 프로그램을 변경하는 예를 나타냅니다.   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // COwnerDraw클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CColorButton m\_button\_blue;  CColorButton m\_button\_red;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  REFLECT\_NOTIFICATIONS()  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_button\_blue.SubclassWindow(GetDlgItem(IDC\_BUTTON\_BLUE));  m\_button\_blue.SetColor(RGB(255, 255, 255), RGB(0, 0, 255));  m\_button\_red.SubclassWindow(GetDlgItem(IDC\_BUTTON\_RED));  m\_button\_red.SetColor(RGB(255, 255, 255), RGB(255, 0, 0));  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "ColorButton.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 CMainDlg 클래스에 CColorButton 클래스의 인스턴스를 멤버 변수로서 2개 선언합니다. 이것들은 OnInitDialog()으로 서브 클래스화 합니다. 서브 클래스화한 후 에SetColor()에 의해 각각 버튼의 문자색과 배경색을 설정합니다. 그리고 메세지 맵에는 메세지 리플렉션을 위해서 REFLECT\_NOTIFICATIONS 매크로를 추가합니다.   　마지막으로 Control.cpp 파일 내에서 ColorButton.h 헤더를 포함 합니다. | |

▣ 컨트룰러 확장 > 커스텀 드로우

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 커스텀 드로우를 이용하면 커스텀 드로우에 대응한 컨트롤의 외관을 커스텀 드로우 보다 간단하게 커스텀 마이즈 할 수가 있습니다. WTL은 커스텀 드로우를 서포트하기 위해서 CCustomDraw클래스를 준비해 있습니다.   　다음에 나타내는 것은 CCustomDraw 클래스를 사용해 스트라이프의 리스트 뷰를 작성하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl11-07/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // CCustomDraw클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>, public CCustomDraw<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CListViewCtrl m\_list\_stripe;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  CHAIN\_MSG\_MAP(CCustomDraw<CMainDlg>) // CCustomDraw클래스에 체인  END\_MSG\_MAP()  DWORD OnPrePaint(int nID, LPNMCUSTOMDRAW lpnmcd){  if(lpnmcd->hdr.idFrom == IDC\_LIST\_STRIPE)  return CDRF\_NOTIFYITEMDRAW;  else  return CDRF\_DODEFAULT;  }  DWORD OnItemPrePaint(int nID, LPNMCUSTOMDRAW lpnmcd){  if(lpnmcd->hdr.idFrom == IDC\_LIST\_STRIPE){  LPNMLVCUSTOMDRAW lpnmlv = (LPNMLVCUSTOMDRAW)lpnmcd;  if(lpnmcd->dwItemSpec % 2){  lpnmlv->clrText = RGB(255, 255, 255);  lpnmlv->clrTextBk = RGB(128, 128, 128);  }  }  return CDRF\_DODEFAULT;  }  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_list\_stripe = GetDlgItem(IDC\_LIST\_STRIPE);  m\_list\_stripe.SetExtendedListViewStyle(LVS\_EX\_INFOTIP | LVS\_EX\_FULLROWSELECT);  CRect rcList;  m\_list\_stripe.GetWindowRect(rcList);  int nScrollWidth = GetSystemMetrics(SM\_CXVSCROLL);  int n3DEdge = GetSystemMetrics(SM\_CXEDGE);  m\_list\_stripe.InsertColumn(0, \_T("이름"), LVCFMT\_LEFT, 190, -1);  m\_list\_stripe.InsertColumn(1, \_T("종류"), LVCFMT\_LEFT,  rcList.Width() - 190 - nScrollWidth - n3DEdge \* 2, -1);  CFindFile find;  if(find.FindFile()){  do{  if(!find.IsDots()){  int nIndex = m\_list\_stripe.GetItemCount();  m\_list\_stripe.AddItem(nIndex, 0, find.GetFileName());  m\_list\_stripe.AddItem(nIndex, 1,  find.IsDirectory() ? \_T("폴더") : \_T("파일"));  }  }while(find.FindNextFile());  }  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 리소스를 작성합니다. 다이얼로그에 리스트 뷰 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 리스트 뷰 컨트롤의 [스타일]에서는 디폴트에 [표시]로 [리포트]를 선택합니다.   |  |  | | --- | --- | | 컨트롤명 | 리소스 ID | | 리스트뷰 | IDC\_LIST\_STRIPE |   　다음에 CCustomDraw 클래스를 사용하기 위해서 stdafx.h 에서 atlctrls.h헤더를 포함 합니다.   　다음으로 CMainDlg 클래스의 base class에 CCustomDraw클래스를 추가해 메세지 맵에 CCustomDraw클래스에 체인을 추가합니다. CCustomDraw 클래스는 내부에 다음과 같은 메세지 맵을 준비해 있습니다.   |  | | --- | | // CCustomDraw클래스  BEGIN\_MSG\_MAP(CCustomDraw< T >)  NOTIFY\_CODE\_HANDLER(NM\_CUSTOMDRAW, OnCustomDraw)  ALT\_MSG\_MAP(1) // 메세지 리플렉션용 대체 메세지 맵  REFLECTED\_NOTIFY\_CODE\_HANDLER(NM\_CUSTOMDRAW, OnCustomDraw)  END\_MSG\_MAP() |   이 메세지 맵에서는 결과적으로OnCustomDraw() 메세지 핸들러를 호출합니다. 이 핸들러에서는 NMCUSTOMDRAW 구조체의 dwDrawStage 멤버의 값에 의해 다음과 같은 멤버 함수를 호출합니다.   |  |  | | --- | --- | | dwDrawStage의 값 | 멤버 함수명 | | CDDS\_PREPAINT | OnPrePaint | | CDDS\_POSTPAINT | OnPostPaint | | CDDS\_PREERASE | OnPreErase | | CDDS\_POSTERASE | OnPostErase | | CDDS\_ITEMPREPAINT | OnItemPrePaint | | CDDS\_ITEMPOSTPAINT | OnItemPostPaint | | CDDS\_ITEMPREERASE | OnItemPreErase | | CDDS\_ITEMPOSTERASE | OnItemPostErase | | CDDS\_ITEMPREPAINT | CDDS\_SUBITEM | OnSubItemPrePaint |   CCustomDraw클래스에의 체인에 의해 커스텀 드로우가 필요한 타이밍에 메시지가 CCustomDraw 클래스에 보내져 이러한 멤버 함수가 실행됩니다. CCustomDraw클래스의 상속 클래스(위의 예에서는CMainDlg클래스) 에서는 이러한 멤버 함수를 오버라이드(override) 하는 것에 의해 컨트롤에 독자적인 외관을 줍니다.   CMainDlg 클래스에서는 우선 OnPrePaint()을 오버라이드(override) 해 컨트롤의 ID IDC\_LIST\_STRIPE의 경우에는 CDRF\_NOTIFYITEMDRAW을 돌려줍니다. 이것에 의해 리스트 뷰의 아이템이 묘화 되는 타이밍에 커스텀 드로우용의 메세지가 보내져 옵니다.   다음에 OnItemPrePaint()를 오버라이드(override) 해 ID를 확인하고 나서 인덱스가 짝수의 아이템의 문자색과 배경색을 설정합니다.   　마지막으로 OnInitDialog() 그리고 리스트 뷰에 아이템을 설정합니다. 컬럼을 2개 추가해 현재 디렉토리의 파일과 폴더를 아이템으로서 리스트 뷰에 추가합니다.   　그런데 CCustomDraw 클래스 내부의 메세지 맵에는 메세지 리플렉션용의 대체 메세지 맵이 준비되어 있습니다. 다음에 나타내는 것은 메세지 리플렉션을 사용한 커스텀 드로우의 방법입니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 메세지 리플렉션에 의한 커스텀 드로우 |
| 메세지 리플렉션을 이용하는 것에 의해 컨트롤 자신에게 커스텀 드로우 시킬 수가 있기때문에 재사용 가능한 컨트롤 클래스를 기술하기 쉬워집니다. 다음에 나타내는 것은 CCustomDraw 클래스와 메세지 리플렉션을 이용해 아이템을 스트라이프 표시하는 리스트 뷰 컨트롤 클래스 CStripeListViewCtrl의 정의하는 예입니다. 이 예에서는 StripeListViewCtrl.h이라고 하는 헤더 파일을 준비해 거기에 CStripeListViewCtrl클래스를 정의하고 있습니다.   |  | | --- | | // StripeListViewCtrl.h  class CStripeListViewCtrl : public CWindowImpl<CStripeListViewCtrl, CListViewCtrl>,  public CCustomDraw<CStripeListViewCtrl>  {  public:  DECLARE\_WND\_SUPERCLASS(\_T("StripeListViewCtrl"), GetWndClassName())  BEGIN\_MSG\_MAP\_EX(CStripeListViewCtrl)  CHAIN\_MSG\_MAP\_ALT(CCustomDraw<CStripeListViewCtrl>, 1)  DEFAULT\_REFLECTION\_HANDLER()  END\_MSG\_MAP()  DWORD OnPrePaint(int nID, LPNMCUSTOMDRAW lpnmcd){  return CDRF\_NOTIFYITEMDRAW;  }  DWORD OnItemPrePaint(int nID, LPNMCUSTOMDRAW lpnmcd){  LPNMLVCUSTOMDRAW lpnmlv = (LPNMLVCUSTOMDRAW)lpnmcd;  if(lpnmcd->dwItemSpec % 2){  lpnmlv->clrText = RGB(255, 255, 255);  lpnmlv->clrTextBk = RGB(128, 128, 128);  }  return CDRF\_DODEFAULT;  }  }; |   이 CStripeListViewCtrl클래스는 CWindowImpl클래스와 CCustomDraw클래스로부터 파생하고 있습니다만 CWindowImpl 클래스의 제2템플릿 인수에는 CListViewCtrl을 지정합니다. 이것에 의해 CStripeListViewCtrl 클래스는 CListViewCtrl의 상속 클래스가 됩니다.   CStripeListViewCtrl 클래스는 반송된 커스텀 드로우용 메세지를 체인에 의해 CCustomDraw 클래스의 대체 메세지 맵에 보냅니다. CCustomDraw 클래스에는 전제와 같이 멤버 함수가 준비되어 있어 CStripeListViewCtrl클래스는 그 중의 OnPrePaint()과 OnItemPrePaint() 를 오버라이드(override) 해 리스트 뷰 아이템을 묘화 하고 있습니다.   　다음에 이 CStripeListViewCtrl클래스를 사용해 앞의 스트라이프 리스트 뷰의 프로그램을 변경하는 예를 나타냅니다.   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h> // CCustomDraw클래스를 사용하기 위해 |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CStripeListViewCtrl m\_list\_stripe;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  REFLECT\_NOTIFICATIONS()  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_list\_stripe.SubclassWindow(GetDlgItem(IDC\_LIST\_STRIPE));  m\_list\_stripe.SetExtendedListViewStyle(LVS\_EX\_INFOTIP | LVS\_EX\_FULLROWSELECT);  CRect rcList;  m\_list\_stripe.GetWindowRect(rcList);  int nScrollWidth = GetSystemMetrics(SM\_CXVSCROLL);  int n3DEdge = GetSystemMetrics(SM\_CXEDGE);  m\_list\_stripe.InsertColumn(0, \_T("이름"), LVCFMT\_LEFT, 190, -1);  m\_list\_stripe.InsertColumn(1, \_T("종류"), LVCFMT\_LEFT,  rcList.Width() - 190 - nScrollWidth - n3DEdge \* 2, -1);  CFindFile find;  if(find.FindFile()){  do{  if(!find.IsDots()){  int nIndex = m\_list\_stripe.GetItemCount();  m\_list\_stripe.AddItem(nIndex, 0, find.GetFileName());  m\_list\_stripe.AddItem(nIndex, 1,  find.IsDirectory() ? \_T("폴더") : \_T("파일"));  }  }while(find.FindNextFile());  }  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "StripeListViewCtrl.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 CMainDlg 클래스에 CStripeListViewCtrl클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것들은 OnInitDialog()으로 서브 클래스화합니다. 서브 클래스화한 후에 리스트 뷰에 아이템을 설정합니다. 그리고 메세지 맵에는 메세지 리플렉션을 위해서 REFLECT\_NOTIFICATIONS 매크로를 추가합니다.   　마지막으로 Control.cpp 파일 내에서 StripeListViewCtrl.h 헤더를 포함 합니다. | |

▣ 확장 컨트룰러 > 비트맵 버튼  
WTL은 표준 컨트롤이나 공용 컨트롤 외에 독자적인 컨트롤 용 클래스를 준비해 있습니다.   
  
　비트맵 버튼은 비트맵 화상으로부터 버튼의 외관을 작성할 수 있는 버튼 컨트롤입니다. WTL은 비트맵 버튼을 작성하기 위해서 CBitmapButton 라고 하는 클래스를 준비해 있습니다. MFC에도 동명의 클래스가 있습니다만 그것과는 사용 방법이 다릅니다.   
  
CBitmapButton 클래스는 CButton 클래스로부터 파생하고 있습니다. 이하에 나타내는 것은 CBitmapButton 클래스를 사용하는 예 입니다. 아래의 그림과 같이 흰 버튼을 작성합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlctrlx.h> // CBitmapButton클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CBitmapButton m\_button\_bmp; // 비트맵 버튼  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_BMP, OnButtonBmp)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  CImageList il;  il.CreateFromImage(IDB\_BITMAP\_BUTTON, 64, 1,  CLR\_NONE, IMAGE\_BITMAP, LR\_CREATEDIBSECTION);  m\_button\_bmp.SubclassWindow(GetDlgItem(IDC\_BUTTON\_BMP));  m\_button\_bmp.SetImageList(il);  m\_button\_bmp.SetImages(0, 1, 2, 3);  m\_button\_bmp.SetToolTipText(\_T("비트 맵 버튼"));  m\_button\_bmp.CenterWindow();  return TRUE;  }  void OnButtonBmp(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("이것은 비트맵 버튼입니다. "));  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 푸쉬 버튼 | IDC\_BUTTON\_BMP |

그리고 프로젝트에 다음과 같은 비트맵 리소스를 IDB\_BITMAP\_BUTTON 이라고 하는 ID으로 추가합니다.   
  
http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl12-01/02.png  
  
이 비트맵은 버튼의 4개 상태(통상 상태 , 눌린 상태 , 포커스를 갖고 있는 상태 , 사용 불가 상태) 를 나타내는 하나의 그림 입니다.   
  
　다음에 stdafx.h 에서는 CBitmapButton 클래스를 사용하기 위해서 atlctrls.h 헤더와 atlctrlx.h 헤더를 포함 합니다. 덧붙여 atlctrls.h 헤더를 먼저 포함 할 필요가 있습니다.   
  
　CMainDlg 클래스에서는 우선 CBitmapButton 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 서브 클래스화 할 필요가 있습니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러에서는 비트맵 리소스(ID:IDB\_BITMAP\_BUTTON)으로부터 이미지 리스트를 작성해 그것을 SetImageList()에 의해 비트맵 버튼으로 설정합니다. 이미지 리스트를 설정한 후는 SetImages()에 의해 이미지 리스트의 인덱스와 버튼 상태를 관련 짓습니다. SetImages()는 제1인수로부터 순서로 통상 상태, 누른 상태, 포커스를 갖고 있는 상태, 사용 불가 상태의 이미지 리스트의 인덱스를 받습니다. 게다가 비트맵 버튼에는 SetToolTipText()에 의해 툴 팁을 설정해 있습니다.   
  
　마지막으로 리소스 ID가 IDC\_BUTTON\_BMP의 커멘드 메세지 핸들러로서 OnButtonBmp()를 추가합니다. 이 핸들러에서는 단지 메시지 박스를 표시하고 있을 뿐입니다.   
  
　이와 같이 MFC의 비트맵 버튼에서는 버튼 상태 별의 4개 비트맵 리소스를 준비해야 하는데 대해 WTL의 비트맵 버튼은 이미지 리스트로부터 버튼의 외관을 작성합니다.   
  
　WTL의 비트맵 버튼은 독자적인 확장 스타일을 준비해 있습니다.

* BMPBTN\_HOVER

마우스 커서가 비트 맵 버튼 위를 지나면 포커스가 맞은 상태의 화상을 표시합니다.

* BMPBTN\_AUTO3D\_SINGLE

화상의 주위에 3D의 경계선을 묘화 합니다.

* BMPBTN\_AUTO3D\_DOUBLE

BMPBTN\_AUTO3D\_SINGLE 보다 조금 굵은 경계선을 묘화 합니다.

* BMPBTN\_AUTOSIZE

화상의 크기에 맞도록 비트맵 버튼의 사이즈를 자동적으로 변경합니다. 이 스타일은 디폴트로 설정되어 있습니다.

* BMPBTN\_SHAREIMAGELISTS

비트맵 버튼의 오브젝트가 파기되어도 비트맵 버튼으로 설정된 이미지 리스트의 오브젝트는 파기하지 않습니다.

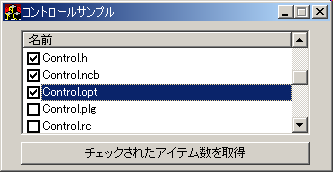
* BMPBTN\_AUTOFIRE

비트맵 버튼을 계속 누르면 반복WM\_COMMAND 메세지를 발생시킵니다.

이러한 확장 스타일은 SetBitmapButtonExtendedStyle()에 의해 설정합니다.

▣ 확장 컨트룰러 > 체크 리스트 뷰

체크 리스트 뷰는 체크 박스 첨부의 리스트 뷰 컨트롤입니다. WTL은 체크 리스트 뷰를 작성하기 위해서 CCheckListViewCtrl라고 하는 클래스를 준비해 있습니다.   
  
CCheckListViewCtrl클래스는 CListViewCtrl클래스로부터 파생하고 있어 디폴트로 LVS\_EX\_CHECKBOXES스타일이 설정되어 있습니다. 이하에 나타내는 것은 CCheckListViewCtrl클래스를 사용하는 예입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlctrlx.h> // CCheckListViewCtrl클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CCheckListViewCtrl m\_list\_dir; // 체크리스트뷰  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_COUNT, OnButtonCount)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_list\_dir.SubclassWindow(GetDlgItem(IDC\_LIST\_DIR));  // 체크리스트뷰에 컬럼 설정  CRect rcList;  m\_list\_dir.GetWindowRect(rcList);  int nScrollWidth = GetSystemMetrics(SM\_CXVSCROLL);  int n3DEdge = GetSystemMetrics(SM\_CXEDGE);  m\_list\_dir.InsertColumn(0, \_T("이름"), LVCFMT\_LEFT,  rcList.Width() - nScrollWidth - n3DEdge \* 2, -1);  // 체크리스트뷰에 아이템 추가  CFindFile find;  if(find.FindFile()){  do{  if(!find.IsDots()){  int nIndex = m\_list\_dir.GetItemCount();  m\_list\_dir.AddItem(nIndex, 0, find.GetFileName());  }  }while(find.FindNextFile());  }  return TRUE;  }  void OnButtonCount(UINT uNotifyCode, int nID, HWND hWndCtl){  int nCheckedCount = 0;  int nCount = m\_list\_dir.GetItemCount();  for(int i=0; i<nCount; i++){  if(m\_list\_dir.GetCheckState(i))  nCheckedCount++;  }  CString strMsg;  strMsg.Format(\_T("체크된 아이템수：%d"), nCheckedCount);  MessageBox(strMsg);  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 리스트 뷰 컨트롤과 버튼 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 리스트 뷰 컨트롤의 [스타일] 에서는 디폴트에 [표시]로 [리포트]를 선택하고 [선택을 항상 표시]에 체크를 넣습니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 리스트뷰 | IDC\_LIST\_DIR |
| 푸쉬 버튼 | IDC\_BUTTON\_COUNT |

　CMainDlg클래스에서는 우선 CCheckListViewCtrl 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 서브 클래스화 할 필요가 있습니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러에서는 체크 리스트 뷰에 현재 디렉토리 내의 폴더와 파일을 아이템으로서 추가합니다.   
  
　마지막으로 리소스 ID가 IDC\_BUTTON\_COUNT의 커멘드 메세지 핸들러로서 OnButtonCount()를 추가합니다. 이 핸들러 에서는 체크 리스트 뷰 중에서 체크되고 있는 아이템을 카운트 해 그것을 메시지 박스를 표시하고 있습니다.   
  
　그런데 위의 예 에서는 체크 리스트 뷰를 작성하기 위해서 다이얼로그 상에 컨트롤 자원을 배치해 서브 클래스화 했습니다만 Create()을 사용해 작성할 수도 있습니다.

|  |
| --- |
| m\_list\_dir.Create(m\_hWnd, CRect(0, 0, 200, 100), NULL, 0, 0, IDC\_LIST\_DIR); |

서브 클래스화에 의해 작성했을 경우는 자원의 프로퍼티에 의해 스타일([리포트]나[선택을 항상 표시]) 을 설정했습니다만 Create()을 사용해 작성했을 경우는 스타일을 제4인수(윈도우 스타일), 제5인수(확장 윈도우 스타일)에 의해 설정합니다. 이러한 인수에 0 를 지정했을 경우는 디폴트의 윈도우 특성으로서 CCheckListViewCtrlTraits 클래스가 사용됩니다. CCheckListViewCtrlTraits 클래스는 atlctrlx.h 헤더 내에서 다음과 같이 선언되고 있습니다.

|  |
| --- |
| template <DWORD t\_dwStyle, DWORD t\_dwExStyle, DWORD t\_dwExListViewStyle>  class CCheckListViewCtrlImplTraits  {  public:  static DWORD GetWndStyle(DWORD dwStyle)  {  return (dwStyle == 0) ? t\_dwStyle : dwStyle;  }  static DWORD GetWndExStyle(DWORD dwExStyle)  {  return (dwExStyle == 0) ? t\_dwExStyle : dwExStyle;  }  static DWORD GetExtendedLVStyle()  {  return t\_dwExListViewStyle;  }  };  typedef CCheckListViewCtrlImplTraits<WS\_CHILD | WS\_VISIBLE |  LVS\_REPORT | LVS\_SHOWSELALWAYS, WS\_EX\_CLIENTEDGE,  LVS\_EX\_CHECKBOXES | LVS\_EX\_FULLROWSELECT> CCheckListViewCtrlTraits; |

CCheckListViewCtrlImplTraits 클래스는 ATL의 CWinTraits 클래스와 달리 윈도우 스타일, 확장 윈도우 스타일에 더해 확장 리스트 뷰 스타일도 설정합니다.   
  
CCheckListViewCtrl클래스는 다음과 같이 정의되고 있습니다.

|  |
| --- |
| // atlctrlx.h  // CCheckListViewCtrl클래스의 base class  template <class T, class TBase = CListViewCtrl,  class TWinTraits = CCheckListViewCtrlTraits>  class ATL\_NO\_VTABLE CCheckListViewCtrlImpl :  public ATL::CWindowImpl<T, TBase, TWinTraits>  {  ...  ...  };  class CCheckListViewCtrl : public CCheckListViewCtrlImpl<CCheckListViewCtrl>  {  public:  DECLARE\_WND\_SUPERCLASS(\_T("WTL\_CheckListView"), GetWndClassName())  }; |

이 때문에 커스텀 마이즈 한 CCheckListViewCtrlImplTraits 클래스를 CCheckListViewCtrlImpl클래스의 제3템플릿 인수로 지정하면 독자적인 스타일의 체크 리스트 뷰 컨트롤을 작성할 수가 있습니다. 다음에 나타내는 것은 확장 리스트 뷰 스타일로서 LVS\_EX\_INFOTIP 스타일을 가진 독자적인 체크 리스트 뷰 컨트롤 클래스를 정의하는 예 입니다.

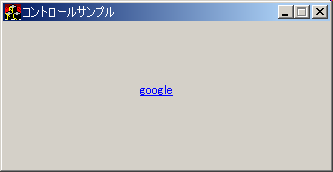
|  |
| --- |
| typedef CCheckListViewCtrlImplTraits<WS\_CHILD | WS\_VISIBLE |  LVS\_REPORT | LVS\_SHOWSELALWAYS, WS\_EX\_CLIENTEDGE,  LVS\_EX\_CHECKBOXES | LVS\_EX\_FULLROWSELECT | LVS\_EX\_INFOTIP> CMyTraits;  class CMyCheckListViewCtrl :  public CCheckListViewCtrlImpl<CMyCheckListViewCtrl, CListViewCtrl, CMyTraits>  {  public:  DECLARE\_WND\_SUPERCLASS(\_T("MyCheckListView"), GetWndClassName())  }; |

덧붙여 LVS\_EX\_INFOTIP와 같은 확장 리스트 뷰 스타일을 변경한다면 SetExtendedListViewStyle()을 사용할 수도 있습니다. 이 때 체크 리스트 뷰 컨트롤로 필수의 LVS\_EX\_CHECKBOXES 스타일도 동시에 지정합니다.

|  |
| --- |
| CCheckListViewCtrl m\_list;  ...  ...  m\_list.SetExtendedListViewStyle(  LVS\_EX\_CHECKBOXES | LVS\_EX\_FULLROWSELECT | LVS\_EX\_INFOTIP); |

▣ 확장 컨트룰러 > 하이퍼 링크

**< 하이퍼 링크 >**하이퍼 링크 컨트롤은 Web 페이지의 하이퍼 링크와 같은 컨트롤입니다. WTL은 하이퍼 링크 컨트롤을 작성하기 위해서 CHyperLink 라고 하는 클래스를 준비해 있습니다.   
  
이하에 나타내는 것은 CHyperLink 클래스를 사용하는 예입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlctrlx.h> // CHyperLink클래스를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CHyperLink m\_link\_google; // 하이퍼 링크  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_link\_google.SubclassWindow(GetDlgItem(IDC\_STATIC\_LINK));  m\_link\_google.SetLabel(\_T("google"));  m\_link\_google.SetHyperLink(\_T("http://www.google.com/"));  return TRUE;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 스태틱 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 스태틱 | IDC\_STATIC\_LINK |

　CMainDlg클래스에서는 우선 CHyperLink 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 서브 클래스화 할 필요가 있습니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러에서는 하이퍼 링크 컨트롤에 라벨과 링크를 설정합니다.   
  
　이와 같이 하이퍼 링크 컨트롤을 작성하기 위해서는 스태틱 컨트롤을 서브 클래스화 합니다. 그리고 Create()를 사용해도 작성할 수 있습니다.   
  
　WTL의 하이퍼 링크 컨트롤은 독자적인 확장 스타일을 준비해 있습니다.

* HLINK\_UNDERLINED

밑줄을 표시합니다. 이 스타일은 디폴트로 설정되어 있습니다.

* HLINK\_NOTUNDERLINED

밑줄을 표시하지 않습니다.

* HLINK\_UNDERLINEHOVER

마우스 커서가 하이퍼 링크 위를 지날 때에 밑줄을 표시합니다.

* HLINK\_COMMANDBUTTON

클릭했을 때(Navigate()가 불려 갔을 때 )에 부모 윈도우에 WM\_COMMAND메세지를 보냅니다. 자동적으로 브라우저를 기동하지 않습니다.

* HLINK\_NOTIFYBUTTON

클릭했을 때(Navigate()가 불려 갔을 때 )에 부모 윈도우에 WM\_NOTIFY 메세지를 보냅니다. 자동적으로 브라우저를 기동하지 않습니다.

* HLINK\_USETAGS

라벨 내의 <A>태그에 둘러싸인 캐릭터 라인을 링크화합니다. 링크 캐릭터 라인은 자동적으로 왼쪽 맞춤 됩니다.

* HLINK\_USETAGSBOLD

라벨 내의 <A>태그에 둘러싸인 캐릭터 라인을 굵은 글씨로 링크화합니다. 링크 캐릭터 라인은 자동적으로 왼쪽 맞춤 됩니다. 이 스타일을 사용하면 HLINK\_UNDERLINED 스타일은 무시됩니다.

* HLINK\_NOTOOLTIP

툴 팁을 표시하지 않습니다.

이러한 확장 스타일은 SetHyperLinkExtendedStyle()에 의해 설정합니다. 덧붙여 이 멤버 함수는 서브 클래스화 하기 전에 호출할 필요가 있습니다. 다음에 나타내는 것은 HLINK\_USETAGS 스타일을 추가해 라벨 내에서 <A>태그를 사용하는 예입니다.



|  |
| --- |
| LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  ... ...    // 컨트롤 설정  m\_link\_google.SetHyperLinkExtendedStyle(HLINK\_USETAGS); // 확장 스타일 설정  m\_link\_google.SubclassWindow(GetDlgItem(IDC\_STATIC\_LINK));  m\_link\_google.SetLabel(\_T("<a>google</a>에 점프"));  m\_link\_google.SetHyperLink(\_T("http://www.google.com/"));  return TRUE;  } |

▣ 확장 컨트룰러 > 모래 시계 커서

**< 모래시계 커서 >**WTL은 윈도우 상의 커서를 간단하게 변경하기 위해서 CWaitCursor 클래스를 준비해 있습니다. 이 클래스는 atlctrlx.h 헤더에 정의되고 있습니다. 다음에 나타내는 것은CWaitCursor 클래스의 가장 간단한 사용 방법입니다.

|  |
| --- |
| // 어느 함수  void function(){  CWaitCursor cursor;  // 시간이 걸리는 처리  } |

CWaitCursor 클래스의 constructor에는 3개의 인수를 지정할 수 있습니다. 제1인수는 인스턴스를 작성한 직후로 커서를 변경하는지 어떤지를 나타내는 bool값으로 true를 지정하면 내부에서Set()라고 하는 멤버 함수가 불려 가자 마자 커서가 바뀝니다. 제2인수는 커서 리소스명, 제3인수는 시스템 커서를 사용하는지 어떤지를 나타내는 bool값입니다. 이러한 인수는 모두 생략 할 수 있어 생략 했을 경우는 디폴트 인수로서 순서에 true,IDC\_WAIT,true가 지정됩니다. 위의 예에서는 constructor인수를 생략 하고 있기 때문에 커서가 모래시계 커서가 됩니다. CWaitCursor 클래스의 소멸자에서는 Restore()이라고 하는 멤버 함수가 불려 가 커서가 원래대로 돌아갑니다.   
  
　Set()나 Restore()는 명시적으로 호출할 수도 있습니다. 다음에 나타내는 것은 전술의 예와 같은 처리를 멤버 함수를 명시적으로 호출해 실행하는 예입니다.

|  |
| --- |
| // 어느 함수  void function(){  CWaitCursor cursor(false);  cursor.Set(); // 커서를 변경  // 시간이 걸리는 처리  cursor.Restore(); // 커서를 바탕으로 되돌리는  } |

이 예에서는 CWaitCursor 클래스의 constructor의 제1인수에 false를 지정해 있기 때문에 인스턴스를 작성한 시점에서는 아직 커서는 변경되지 않습니다. Set()을 호출했을 때에 처음으로 커서가 변경됩니다.   
  
　다음에 나타내는 것은 독자적인 커서를 표시하는 예입니다. 덧붙여 미리 프로젝트에 IDC\_CURSOR1라고 하는 ID의 커서 리소스를 추가해 둡니다.

|  |
| --- |
| // 어느 함수  void function(){  CWaitCursor cursor(true, MAKEINTRESOURCE(IDC\_CURSOR1), false);  // 있는 처리  } |

이 예에서는 시스템 커서가 아니고 독자적인 커서 자원을 사용하기 위해 constructor의 제3인수에는 false를 지정합니다.   
  
　덧붙여 WTL는 독자적인 커서를 표시하기 위해서 CWaitCursor클래스로부터 파생한 CCustomWaitCursor 클래스를 준비해 있습니다. 다음에 나타내는 것은 CCustomWaitCursor클래스를 사용하는 예입니다.

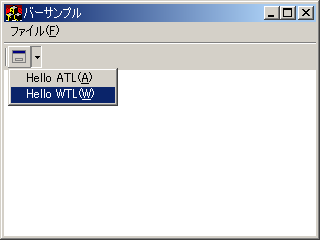
|  |
| --- |
| // 어느 함수  void function(){  CCustomWaitCursor cursor(IDC\_CURSOR1);  // 있는 처리  } |

CCustomWaitCursor클래스의 constructor의 제1인수에는 커서 리소스의 ID를 지정합니다. 제2, 3인수에는 각각 인스턴스를 작성한 직후로 커서를 변경하는지 어떤지를 나타내는 bool 값과 커서 자원이 들어간 모듈의 인스턴스 핸들을 지정할 수 있습니다만 어느 쪽이나 생략 가능합니다. 생략 했을 경우는 디폴트 인수로서 각각 true,NULL이 지정됩니다.

**▣ 바 > 툴바**WTL에서 공용 컨트롤의 툴바 컨트롤을 CToolBarCtrlT 라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CToolBarCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CToolBarCtrlT<ATL::CWindow> CToolBarCtrl; |

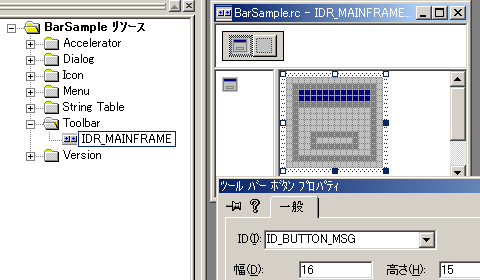
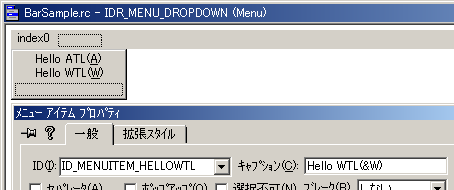
이것은 CToolBarCtrl클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CToolBarCtrl클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CToolBarCtrl클래스를 사용하는 예입니다. 프레임 윈도우를 작성해 툴바를 추가해 툴바 위의 최초의 버튼을 드롭 다운 스타일로 합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // mainfrm.h내  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateToolBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  NOTIFY\_CODE\_HANDLER\_EX(TBN\_DROPDOWN, OnToolbarDropdown)  COMMAND\_ID\_HANDLER\_EX(ID\_BUTTON\_MSG, OnButtonMsg)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLOATL, OnMenuHelloATL)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_HELLOWTL, OnMenuHelloWTL)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 툴바를 작성  CreateSimpleToolBar();  UIAddToolBar(m\_hWndToolBar);  // 툴바의 스타일 변경  CToolBarCtrl bar = m\_hWndToolBar;  bar.SetStyle(bar.GetStyle() | TBSTYLE\_FLAT);  bar.SetExtendedStyle(TBSTYLE\_EX\_DRAWDDARROWS);  // 툴바상의 버튼의 스타일을 변경  TBBUTTONINFO bi = {sizeof(bi), TBIF\_STYLE};  bar.GetButtonInfo(ID\_BUTTON\_MSG, &bi);  bi.fsStyle |= TBSTYLE\_DROPDOWN;  bar.SetButtonInfo(ID\_BUTTON\_MSG, &bi);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  LRESULT OnToolbarDropdown(LPNMHDR pnmh){  LPNMTOOLBAR pnmtb = (LPNMTOOLBAR)pnmh;  if(pnmtb->iItem == ID\_BUTTON\_MSG){  CToolBarCtrl bar = pnmtb->hdr.hwndFrom;  // 드롭 다운 메뉴를 표시하는 위치를 취득  CRect rcButton;  bar.GetRect(ID\_BUTTON\_MSG, rcButton);  bar.MapWindowPoints(HWND\_DESKTOP, rcButton);  TPMPARAMS tpm = {sizeof(tpm)};  tpm.rcExclude = rcButton;  // 드롭 다운 메뉴 표시  CMenu menuDropdown;  menuDropdown.LoadMenu(IDR\_MENU\_DROPDOWN);  menuDropdown.GetSubMenu(0).TrackPopupMenuEx(  TPM\_LEFTALIGN | TPM\_LEFTBUTTON | TPM\_VERTICAL,  rcButton.left, rcButton.bottom, m\_hWnd, &tpm);  }  return TBDDRET\_DEFAULT;  }  void OnButtonMsg(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello World"));  }  void OnMenuHelloATL(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello ATL"));  }  void OnMenuHelloWTL(UINT uNotifyCode, int nID, HWND hWndCtl){  MessageBox(\_T("Hello WTL"));  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

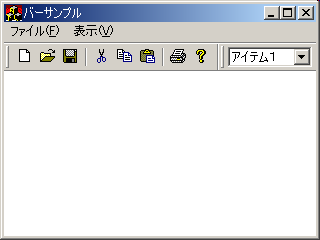
　우선 프로젝트에 IDR\_MAINFRAME 라고 하는 ID의 툴바 리소스를 추가하고 다음과 같은 버튼을 추가합니다. 버튼 ID는 ID\_BUTTON\_MSG으로 합니다.   
  
  
  
게다가 드롭 다운용의 메뉴 자원을 추가합니다. 메뉴 리소스 ID는 IDR\_MENU\_DROPDOWN으로 하고 톱 레벨에 [index0]를 그 아래에 [Hello ATL]메뉴 아이템과 [Hello WTL] 메뉴 아이템을 추가합니다. [Hello ATL] 메뉴 아이템에는ID\_MENUITEM\_HELLOATL, [Hello WTL] 메뉴 아이템에는ID\_MENUITEM\_HELLOWTL 라고 하는ID을 설정합니다.   
  
  
  
　다음으로 stdafx.h 에서는 CToolBarCtrl 클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   
  
　CMainFrame클래스에서는 일반적인 프레임 윈도우를 작성합니다. WM\_CREATE메세지 핸들러로 우선 CreateSimpleToolBar()을 호출해 툴바를 작성합니다. 이것에 의해 프레임 윈도우의 멤버 변수인 m\_hWndToolBar에 툴바의 핸들이 설정됩니다. 다음에 CToolBarCtrl 클래스의 인스턴스를 이 툴바의 핸들로 초기화 하고 툴바의 스타일을 변경합니다. 여기에서는 툴바 상의 드롭 다운 버튼에 화살표를 추가하기 위해 TBSTYLE\_EX\_DRAWDDARROWS 스타일을 설정합니다. 그리고 ID가 ID\_BUTTON\_MSG의 버튼에 드롭 다운 스타일을 추가합니다.   
  
　다음으로 통지 코드가 TBN\_DROPDOWN의 통지 메세지 핸들러를 추가합니다. 이 핸들러는 드롭 다운 버튼의 화살표 부분을 클릭할 때마다 불려 갑니다. 여기에서는 버튼 ID를 확인하고 드롭 다운 메뉴를 표시하고 있습니다.   
  
　다음으로 커멘드 메세지 핸들러를 추가합니다. OnButtonMsg()는 드롭 다운 버튼을 눌렀을 때에 불려지는 OnMenuHelloATL()와OnMenuHelloWTL()는 드롭 다운 메뉴 항목을 실행했을 때에 불려 집니다. 이러한 핸들러 에서는 메시지 박스를 표시하고 있을 뿐입니다.   
  
　마지막으로 \_tWinMain() 에서 ICC\_WIN95\_CLASSES 플래그를 지정해 AtlInitCommonControls()를 호출합니다. 이것에 의해 툴바 컨트롤이 사용 가능하게 됩니다.

**▣ 바 > 리바**

WTL에서 공용 컨트롤의 리바 컨트롤을 CReBarCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CReBarCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlctrls.h  typedef CReBarCtrlT<ATL::CWindow> CReBarCtrl; |

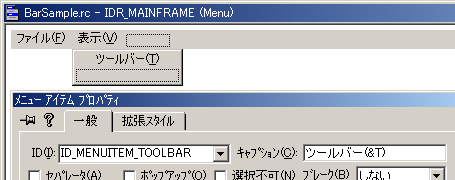
이것은 CReBarCtrl클래스는 CWindow클래스의 상속 클래스인 것을 의미합니다.   
  
　WTL의 CReBarCtrl클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CReBarCtrl클래스를 사용하는 예입니다. 프레임 윈도우를 작성해 리바를 추가하고 그 리바의 밴드에 툴바와 combobox 컨트롤을 추가합니다. 리바의 밴드에 추가된 툴바는 표시/비표시를 변환되도록 합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  enum { IDC\_COMBO\_REBAR = 1001 };  CComboBox m\_combo\_rebar;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateToolBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  UPDATE\_ELEMENT(ID\_MENUITEM\_TOOLBAR, UPDUI\_MENUPOPUP)  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOOLBAR, OnMenuToolbar)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 리버를 작성  CreateSimpleReBar();  // 툴바를 작성해 밴드에 추가  HWND hToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  AddSimpleReBarBand(hToolBar);  UIAddToolBar(hToolBar);  UISetCheck(ID\_MENUITEM\_TOOLBAR, 1);  // combobox를 작성해 밴드에 추가  m\_combo\_rebar.Create(m\_hWnd, CRect(0, 0, 100, 100), NULL,  WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | CBS\_DROPDOWNLIST, 0, IDC\_COMBO\_REBAR);  m\_combo\_rebar.SetFont(AtlGetDefaultGuiFont());  m\_combo\_rebar.AddString(\_T("아이템 1"));  m\_combo\_rebar.AddString(\_T("아이템 2"));  AddSimpleReBarBand(m\_combo\_rebar);  // 리버의 사이즈를 조정  SizeSimpleReBarBands();  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuToolbar(UINT uNotifyCode, int nID, HWND hWndCtl){  static BOOL bVisible = TRUE;  bVisible = !bVisible;  CReBarCtrl rebar = m\_hWndToolBar;  int nBandIndex = rebar.IdToIndex(ATL\_IDW\_BAND\_FIRST);  rebar.ShowBand(nBandIndex, bVisible);  UISetCheck(ID\_MENUITEM\_TOOLBAR, bVisible);  UpdateLayout();  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 도구 모음의 톱 레벨에 [표시]를 작성하고 그 아래에 [툴바] 메뉴 아이템을 추가합니다. 이 [툴바]에는 ID\_MENUITEM\_TOOLBAR 라고 하는 ID를 설정합니다.   
  
  
  
　다음으로 stdafx.h 에서는 CReBarCtrl 클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   
  
　CMainFrame 클래스에서는 일반적인 프레임 윈도우를 작성합니다. WM\_CREATE 메세지 핸들러에서는 우선 CreateSimpleReBar()을 호출해 리바를 작성합니다. 이것에 의해 프레임 윈도우의 멤버 변수인 m\_hWndToolBar에 리바의 핸들이 설정됩니다. 다음에 툴바 컨트롤과 combobox 컨트롤을 작성하고 각각 리버의 밴드에 추가합니다.   
  
　다음으로 메뉴 아이템 [툴바](ID：ID\_MENUITEM\_TOOLBAR) 용의 커멘드 메세지 핸들러를 추가합니다. 이 핸들러에서는 정적 변수 bVisible에 의해 툴바의 가시 상태를 보관 유지하며 CReBarCtrl 클래스의 멤버 함수인 ShowBand()로 가시 상태를 새로 바꾸고 있습니다. 덧붙여 이 핸들러의 코드는 ATL/WTL AppWizard 이 출력하는 것과 거의 같습니다.   
  
　마지막으로 \_tWinMain() 에서 ICC\_COOL\_CLASSES 플래그를 지정해 AtlInitCommonControls()를 호출합니다. 이것에 의해 리바 컨트롤이 사용 가능하게 됩니다.

▣ 바 > 스테터스 바

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WTL에서 공용 컨트롤의 상태 바 컨트롤을 CStatusBarCtrlT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CStatusBarCtrlT 클래스의 base class로서 사용됩니다. atlctrls.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlctrls.h  typedef CStatusBarCtrlT<ATL::CWindow> CStatusBarCtrl; |   이것은 CStatusBarCtrl 클래스는 CWindow 클래스의 상속 클래스인 것을 의미합니다.   　WTL의 CStatusBarCtrl클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CStatusBarCtrl클래스를 사용하는 예입니다. 프레임 윈도우를 작성해 상태 바를 추가하고 그 상태 바의 2번째 의 페인에 프레임 윈도우의 클라이언트 영역상의 마우스 좌표를 표시합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl13-03/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |  |  | | --- | | // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_MOUSEMOVE(OnMouseMove)  MSG\_WM\_SIZE(OnSize)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnSize(UINT uType, CSize size){  // 마우스 좌표를 표시하는 페인의 폭  int cxPosPane = 90;  // 데포르트페인의 폭  CRect rcClient;  GetClientRect(rcClient);  int cxDefaultPane = rcClient.right - cxPosPane  - ::GetSystemMetrics(SM\_CXVSCROLL) - ::GetSystemMetrics(SM\_CXEDGE);  // 상태 바에 페인을 설정  CStatusBarCtrl bar = m\_hWndStatusBar;  int nPanes[] = {cxDefaultPane, cxDefaultPane + cxPosPane};  bar.SetParts(sizeof(nPanes)/sizeof(nPanes[0]), nPanes);  // base class의WM\_SIZE메세지 핸들러도 호출하기 (위해)때문에  SetMsgHandled(false);  }  void OnMouseMove(UINT uFlags, CPoint ptClient){  CString strPos;  strPos.Format(\_T("X:%d, Y:%d"), ptClient.x, ptClient.y);  CStatusBarCtrl bar = m\_hWndStatusBar;  bar.SetText(1, strPos);  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 stdafx.h 에서는 CStatusBarCtrl클래스를 사용하기 위해서 atlctrls.h 헤더를 포함 합니다.   　CMainFrame 클래스에서는 일반적인 프레임 윈도우를 작성해 WM\_CREATE 메세지 핸들러로 CreateSimpleStatusBar()를 호출해 상태 바를 작성합니다. 이것에 의해 프레임 윈도우의 멤버 변수인 m\_hWndStatusBar에 상태 바의 핸들이 설정됩니다.   　다음에 WM\_SIZE 메세지 핸들러를 추가하고 상태 바에 2개의 페인을 추가합니다. 1번째 는 프레임 윈도우의 크기에 맞추어 폭을 조정하는 데포르트 페인, 2번째 는 프레임 윈도우의 클라이언트 영역상의 마우스 좌표를 표시하는 페인입니다. 페인을 설정하려면 SetParts()를 호출합니다. 제1인수에는 페인의 수, 제2인수에는 페인의 우단의 좌표를 요소로 하는 배열에의 포인터를 지정합니다. 덧붙여 WM\_SIZE 메세지는 base class인 CFrameWindowImpl클래스내에서도 매핑 되고 있기 때문에 마지막에 SetMsgHandled(false)를 호출할 필요가 있습니다.   　다음으로 WM\_MOUSEMOVE 메세지 핸들러를 추가합니다. 여기에서는 클라이언트 좌표를 SetText()에 의해 상태 바의 2번째의 페인(인덱스는1) 에 표시하고 있습니다.   　마지막으로 \_tWinMain()안에서 ICC\_WIN95\_CLASSES 플래그를 지정해 AtlInitCommonControls()를 호출합니다. 이것에 의해 상태 바 컨트롤이 사용 가능하게 됩니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 멀티 페인 스테이터스 바 |
| 위의 예에서는 상태 바에 복수의 페인(멀티 페인) 을 설정하기 위해서 WM\_SIZE 메세지 핸들러를 추가해 SetParts()를 호출했습니다. WTL은 이러한 작업을 간략화하기 위해서 CMultiPaneStatusBarCtrl라고 하는 클래스를 준비해 있습니다. CMultiPaneStatusBarCtrl 클래스는 CStatusBarCtrl 클래스의 상속 클래스로 atlctrlx.h 헤더에서는 다음과 같이 정의되고 있습니다.   |  | | --- | | // atlctrlx.h  template <class T, class TBase = CStatusBarCtrl>  class ATL\_NO\_VTABLE CMultiPaneStatusBarCtrlImpl : public ATL::CWindowImpl< T, TBase >  {  ... ...  };  class CMultiPaneStatusBarCtrl  : public CMultiPaneStatusBarCtrlImpl<CMultiPaneStatusBarCtrl>  {  public:  DECLARE\_WND\_SUPERCLASS(\_T("WTL\_MultiPaneStatusBar"), GetWndClassName())  }; |   　다음에 나타내는 것은 전술의 프로그램을 CMultiPaneStatusBarCtrl클래스를 사용해 고쳐 쓰는 예입니다.   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlctrlx.h> // CMultiPaneStatusBarCtrl클래스를 사용하기 위해  #include <atlframe.h> |  |  | | --- | | // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CMultiPaneStatusBarCtrl m\_wndStatusBar;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_MOUSEMOVE(OnMouseMove)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  m\_hWndStatusBar = m\_wndStatusBar.Create(m\_hWnd);  UIAddStatusBar(m\_hWndStatusBar);  // 상태 바에 페인을 설정  int nPanes[] = {ID\_DEFAULT\_PANE, IDS\_PANE\_POS};  m\_wndStatusBar.SetPanes(nPanes, sizeof(nPanes)/sizeof(nPanes[0]));  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMouseMove(UINT uFlags, CPoint ptClient){  CString strPos;  strPos.Format(\_T("X:%d, Y:%d"), ptClient.x, ptClient.y);  m\_wndStatusBar.SetPaneText(IDS\_PANE\_POS, strPos);  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // Control.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 stdafx.h 에서는 CMultiPaneStatusBarCtrl클래스를 사용하기 위해서 atlctrlx.h 헤더를 인클루드 합니다.   　CMainFrame 클래스에서는 CMultiPaneStatusBarCtrl 클래스의 인스턴스를 멤버 변수로서 선언하고 WM\_CREATE 메세지 핸들러로 Create() 를 호출하는 것에 의해 멀티 페인 스테이터스 바를 작성합니다. Create()는 반환 값 으로서 상태 바의 핸들을 돌려주므로 그것을 프레임 윈도우의 멤버 변수인 m\_hWndStatusBar에 대입합니다.   덧붙여 Create()에는 2개의 버젼이 있어 한쪽은 제2인수에 캐릭터 라인을 지정하는 것, 또 한쪽은 제2인수에 캐릭터 라인 리소스 ID를 지정하는 것입니다. 제2인수를 생략 했을 경우는 후자의 Create()가 불려져 디폴트의 캐릭터 라인 리소스 ID으로서 ATL\_IDS\_IDLEMESSAGE가 지정됩니다. 양 버젼 모두 제1인수에 부모 윈도우 핸들을 지정하며 제3인수에는 스타일, 제4인수에는 ID를 지정할 수 있습니다. 제3인수를 생략 했을 경우는 디폴트 스타일로서 WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS | SBARS\_SIZEGRIP 가 지정되어 제4인수를 생략 했을 경우는 디폴트ID로서 ATL\_IDW\_STATUS\_BAR가 지정됩니다.   　다음으로 SetPanes()를 호출해 상태 바에 페인을 설정합니다. 제1인수에는 캐릭터 라인 리소스 ID를 요소로 한 배열에의 포인터를 지정하고 제2인수에는 페인의 수를 지정합니다. SetPanes()는 배열내의 캐릭터 라인 리소스 ID 로부터 캐릭터 라인을 취득해 그 캐릭터 라인의 폭과 같은 크기의 페인을 작성합니다. 여기에서는 마우스 좌표를 표시하기 위해서 ID가 IDS\_PANE\_POS 라고 하는 캐릭터 라인 자원을 준비해 캡션으로서 「X: 0000, Y: 0000」을 설정합니다. 이것은 폭과 높이가 각각 4자리수까지 표시할 수 있는 사이즈의 페인을 작성하는 것을 의미합니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl13-03/02.png  덧붙여 배열의 1번째로 지정해 ID\_DEFAULT\_PANE는 WTL이 준비해 있는 ID로 프레임 윈도우의 사이즈가 변경될 때마다 거기에 맞추어 크기가 조정되는 페인을 의미합니다.   　다음에 WM\_MOUSEMOVE 메세지 핸들러에서는 SetPaneText에 의해 상태 바의 ID가 IDS\_PANE\_POS의 페인에 클라이언트 좌표를 표시하고 있습니다.   　이 예에서는 WM\_SIZE 메세지 핸들러를 작성하지 않습니다. 이것은 CMultiPaneStatusBarCtrl클래스 내에는 WM\_SIZE 메세지 핸들러가 있으며 프레임 윈도우의 사이즈를 변경할 때마다 상태 바의 페인이 자동적으로 조절되기 때문입니다. | |

▣ 바 > 커맨드 바

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **< 커맨드 바 >**  커맨드 바란 InternetExplorer의 도구 모음과 같은 그립이 붙어 있어 이동 가능한 바입니다. WTL은 커맨드 바를 작성하기 위해서 CCommandBarCtrl클래스를 준비해 있습니다. CCommandBarCtrl클래스를 사용하면 메뉴 리소스를 준비하는 것 만으로 간단하게 커맨드 바를 작성할 수가 있습니다. 그리고 화상 리소스를 준비하면 메뉴 아이템에 화상을 관련 지을 수가 있습니다. (덧붙여 화상은 CCommandBarCtrl클래스의 멤버 함수인 SetImagesVisible(false)을 호출하는 것에 의해 비 표시로 할 수도 있습니다. )  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl13-04/01.png  게다가 커맨드 바의 표시 폭이 부족한 경우에서도 자동적으로 셰브론(**>>**마크) 이 유효하게 됩니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl13-04/02.png  물론 보통 도구 모음과 같이 상태 바에 플라이바 텍스트 (메뉴 아이템상에 커서를 두었을 때에 상태 바에 표시되는 텍스트) 를 표시할 수가 있습니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl13-04/03.png  CCommandBarCtrl 클래스는 CToolBarCtrl클래스의 상속 클래스입니다. WTL에서 도구모음 으로 가장한 툴바를 리버의 밴드에 추가하는 것에 의해 커맨드 바를 작성합니다.   　이하에 나타내는 것은 CCommandBarCtrl클래스를 사용하는 예입니다. 프레임 윈도우를 작성해 커맨드 바를 추가합니다. 덧붙여 설명의 형편상 툴바와 상태 바도 추가해 툴바는 표시/비표시를 변환되도록 합니다.   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlctrlw.h> // CCommandBarCtrl클래스를 사용하기 위해  #include <atlframe.h> |  |  | | --- | | // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CCommandBarCtrl m\_wndCmdBar;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateToolBar();  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  UPDATE\_ELEMENT(ID\_MENUITEM\_TOOLBAR, UPDUI\_MENUPOPUP)  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_TOOLBAR, OnMenuToolbar)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 코맨드 바를 작성  HWND hWndCmdBar = m\_wndCmdBar.Create(m\_hWnd,  rcDefault, NULL, ATL\_SIMPLE\_CMDBAR\_PANE\_STYLE);  // 기존의 도구모음을 아탓치  m\_wndCmdBar.AttachMenu(GetMenu());  // 메뉴 아이템에 관련지을 수 있었던 툴바의 화상을 로드  m\_wndCmdBar.LoadImages(IDR\_MAINFRAME);  // 기존의 도구모음을 삭제  SetMenu(NULL);  // 툴바를 작성  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  UIAddToolBar(hWndToolBar);  // 리버를 작성해 , 코맨드 바와 툴바를 밴드에 추가  CreateSimpleReBar(ATL\_SIMPLE\_REBAR\_NOBORDER\_STYLE);  AddSimpleReBarBand(hWndCmdBar);  AddSimpleReBarBand(hWndToolBar, NULL, TRUE);  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  UISetCheck(ID\_MENUITEM\_TOOLBAR, true);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuToolbar(UINT uNotifyCode, int nID, HWND hWndCtl){  static BOOL bVisible = TRUE;  bVisible = !bVisible;  CReBarCtrl rebar = m\_hWndToolBar;  int nBandIndex = rebar.IdToIndex(ATL\_IDW\_BAND\_FIRST + 1);  rebar.ShowBand(nBandIndex, bVisible);  UISetCheck(ID\_MENUITEM\_TOOLBAR, bVisible);  UpdateLayout();  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 도구 모음 자원의 톱 레벨에 [표시]를 작성해 그 아래에 [툴바] 메뉴 아이템을 추가합니다. 이 [툴바]에는ID\_MENUITEM\_TOOLBAR 이라고 하는ID을 설정합니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl13-04/04.png  　다음에 stdafx.h 에서는 CCommandBarCtrl클래스를 사용하기 위해서 atlctrlw.h 헤더를 포함 합니다.   　CMainFrame 클래스에서는 일반적인 프레임 윈도우를 작성합니다. CMainFrame클래스의 멤버 변수로서 CCommandBarCtrl 클래스의 인스턴스를 선언해 WM\_CREATE 메세지 핸들러로 Create() 를 호출하는 것에 의해 커맨드 바를 작성합니다. 이Create()의 인수의 의미는 CWindowImpl::Create()과 같아 여기에서는 윈도우 스타일에 ATL\_SIMPLE\_CMDBAR\_PANE\_STYLE를 지정해 있습니다. 이 스타일은 atlctrlw.h 헤더 내에서 다음과 같이 정의되고 있습니다. 덧붙여 표시상의 사정으로 개행 위치등을 변경하고 있습니다.   |  | | --- | | // atlctrlw.h  // Window Styles:  #define CBRWS\_TOP CCS\_TOP  #define CBRWS\_BOTTOM CCS\_BOTTOM  #define CBRWS\_NORESIZE CCS\_NORESIZE  #define CBRWS\_NOPARENTALIGN CCS\_NOPARENTALIGN  #define CBRWS\_NODIVIDER CCS\_NODIVIDER  // standard command bar styles  #define ATL\_SIMPLE\_CMDBAR\_PANE\_STYLE \  (WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS | \  CBRWS\_NODIVIDER | CBRWS\_NORESIZE | CBRWS\_NOPARENTALIGN) |   커맨드 바를 작성한 후는 커맨드 바에 메뉴 아이템을 추가합니다. 위의 예에서는AttachMenu()을 호출하는 것에 의해 프레임 윈도우의 도구 모음으로부터 메뉴 아이템을 읽어들이고 있습니다만 LoadMenu()을 호출해 메뉴 리소스로부터 직접 아이템을 로드할 수도 있습니다.   |  | | --- | | // 코맨드 바를 작성  HWND hWndCmdBar = m\_wndCmdBar.Create(m\_hWnd,  rcDefault, NULL, ATL\_SIMPLE\_CMDBAR\_PANE\_STYLE);  // 메뉴 자원을 로드  m\_wndCmdBar.LoadMenu(IDR\_MAINFRAME); |   다음에 LoadImages()를 호출하는 것에 의해 툴바의 화상을 로드하고 있습니다. 이것에 의해 ID가 동일한 메뉴 아이템과 툴바의 화상을 관련지을 수 있어 메뉴 아이템의 옆에 화상이 표시되게 됩니다. 덧붙여 메뉴 아이템의 옆에 화상을 표시하기 위해서는 AddBitmap()이나AddIcon()를 사용할 수도 있습니다.   다음으로 툴바와 리버를 작성해 밴드에 커맨드 바와 툴바를 추가합니다. 툴바는 커맨드 바 아래에 추가하기 때문에 AddSimpleReBarBand의 제3인수에는TRUE를 지정합니다.   　마지막으로 메뉴 아이템 [툴바](ID：ID\_MENUITEM\_TOOLBAR) 용의 커멘드 메세지 핸들러를 추가합니다. 이 핸들러에서는 정적 변수 bVisible에 의해 툴바의 가시 상태를 보관 유지해 CReBarCtrl클래스의 멤버 함수인 ShowBand()으로 가시 상태를 새로 바꾸고 있습니다. 덧붙여 이 핸들러의 코드는 ATL/WTL AppWizard 이 출력하는 것과 거의 같습니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 커맨드 바 스타일의 pop-up menu |
| 통상의 메뉴가 CMenu::TrackPopupMenu()를 호출하는 것에 의해 pop-up 할 수 있도록 커맨드 바 스타일의 메뉴도 CCommandBarCtrl::TrackPopupMenu()를 호출하는 것에 의해 pop-up 할 수 있습니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl13-04/05.png  이하에 나타내는 것은 커맨드 바 스타일의 pop-up menu를 표시하는 코드의 예입니다. 전제의 프로그램에 이하의 코드를 추가하는 것으로써 클라이언트 영역을 오른쪽 클릭했을 때에 인덱스 0 의 커맨드 바 메뉴 아이템( [파일] 이하 ) 이 pop-up menu로서 표시됩니다.   |  | | --- | | // mainfrm.h  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  ...  ...  END\_MSG\_MAP()  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 클라이언트 영역의 좌상으로 설정  if(pt.x == -1 && pt.y == -1){  pt.SetPoint(0, 0);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  // 코맨드 바의 메뉴 핸들을 취득  CMenuHandle menu = m\_wndCmdBar.GetMenu();  // 인덱스0의 메뉴 아이템을 pop-up menu로서 표시  m\_wndCmdBar.TrackPopupMenu(menu.GetSubMenu(0),  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, NULL);  }else{  SetMsgHandled(false);  }  } |   이와 같이 pop-up menu를 커맨드 바 스타일로 표시하기 위해서는 CCommandBarCtrl::TrackPopupMenu()의 제1인수에 표시하는 메뉴의 핸들을 지정합니다. | |

▣ 공용 다이얼로그 > 파일 선택 다이얼로그

**< 파일 선택 다이얼로그 >**  
  
WTL은 common dialog의 파일 선택 다이얼로그를 CFileDialogImpl이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 이 클래스는 ATL의 CDialogImpl 클래스와 같이 base class로서의 보고 사용합니다. atldlgs.h 헤더에서는 CFileDialogImpl 클래스의 상속 클래스로서 CFileDialog 클래스가 정의되고 있습니다.

|  |
| --- |
| // atldlgs.h  class CFileDialog : public CFileDialogImpl<CFileDialog>  {  ...  ... |

CFileDialog 클래스는 [파일을 여는] 다이얼로그와 [이름을 붙여 보존] 다이얼로그를 서포트하고 있습니다.   
  
　WTL의 CFileDialog 클래스는 MFC의 동명의 클래스와 동등의 기능을 갖추고 있습니다. 이하에 나타내는 것은 CFileDialog 클래스를 사용하는 예 입니다. 덧붙여 WTL의 common dialog 클래스를 사용하기 위해서는 atldlgs.h 헤더를 포함 할 필요가 있습니다.



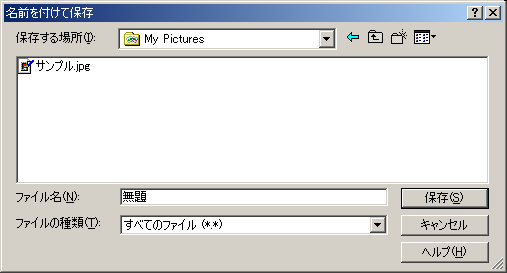
|  |
| --- |
| // [파일을 여는]다이얼로그  CFileDialog dlg(TRUE, \_T("txt"), NULL, OFN\_HIDEREADONLY | OFN\_CREATEPROMPT,  \_T("텍스트 파일 (\*.txt)\0\*.txt\0모든 파일 (\*.\*)\0\*.\*\0\0"));  if(dlg.DoModal() == IDOK){  CString strMsg;  strMsg.Format(\_T("파일명：%s\n파스：%s"), dlg.m\_szFileTitle, dlg.m\_szFileName);  MessageBox(strMsg);  } |

|  |
| --- |
| // [이름을 붙여 보존]다이얼로그  CFileDialog dlg(FALSE, \_T("txt"), \_T("무제"), OFN\_OVERWRITEPROMPT,  \_T("텍스트 파일 (\*.txt)\0\*.txt\0모든 파일 (\*.\*)\0\*.\*\0\0"));  if(dlg.DoModal() == IDOK){  CString strMsg;  strMsg.Format(\_T("파일명：%s\n패스：%s"), dlg.m\_szFileTitle, dlg.m\_szFileName);  MessageBox(strMsg);  } |

CFileDialog 클래스의 constructor의 제1인수에는BOOL값을 지정합니다. TRUE의 경우는 [파일을 여는] 다이얼로그를 작성하며, FALSE의 경우는 [이름을 붙여 보존] 다이얼로그를 작성합니다. 나머지의 인수는 생략 가능하며 순서에 디폴트 확장자(extension), 디폴트 파일명 , 플래그 , 필터 , 부모 윈도우의 핸들입니다.   
  
파일 선택 다이얼로그를 표시하기 위해서는 DoModal()을 사용합니다. [여는](또는[보존]) 버튼을 눌러 파일 선택 다이얼로그를 다물었을 경우는 반환 값으로서 IDOK가, [캔슬]을 눌러 닫았을 경우는IDCANCEL이 되돌아 옵니다.   
  
IDOK해서 되돌아 왔을 경우는 CFileDialog 클래스의 멤버인 m\_szFileTitle과m\_szFileName에 각각 파일명과 파일 패스가 설정되어 있습니다.   
  
CFileDialog 클래스의 base class인 CFileDialogImpl클래스는 메세지 맵을 준비해 있습니다. 이것은 파일 선택 다이얼로그용의 WM\_NOTIFY 메세지를 매핑 합니다. 메세지 맵은 WM\_NOTIFY 메세지의 통지 코드에 의해 간접적으로 다음과 같은 멤버 함수를 호출합니다.

|  |  |
| --- | --- |
| 통지 코드 | 멤버 함수명 |
| CDN\_FILEOK | OnFileOK |
| CDN\_FOLDERCHANGE | OnFolderChange |
| CDN\_HELP | OnHelp |
| CDN\_INITDONE | OnInitDone |
| CDN\_SELCHANGE | OnSelChange |
| CDN\_SHAREVIOLATION | OnShareViolation |
| CDN\_TYPECHANGE | OnTypeChange |

CFileDialogImpl 클래스의 상속 클래스는 이러한 멤버 함수를 오버라이드(override) 하는 것에 의해 파일 선택 다이얼로그의 동작을 커스텀 마이즈 할 수가 있습니다.   
  
다음에 나타내는 것은 OnInitDone()이라는 OnHelp()를 오버라이드(override) 하여 독자적인 파일 선택 다이얼로그를 작성하는 예입니다. 이 파일 선택 다이얼로그에는 [헬프] 버튼이 추가되어 부모 윈도우의 중앙에 표시됩니다. 이 예에서는 CustomDlg.h이라고 하는 헤더 파일을 준비하고 거기에 CCustomDlg클래스를 정의하고 있습니다.



|  |
| --- |
| // CustomDlg.h  class CCustomDlg : public CFileDialogImpl<CCustomDlg>  {  public:  // constructor  CCustomDlg(BOOL bOpenFileDialog,  LPCTSTR lpszDefExt = NULL,  LPCTSTR lpszFileName = NULL,  DWORD dwFlags = OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT,  LPCTSTR lpszFilter = NULL,  HWND hWndParent = NULL)  : CFileDialogImpl<CCustomDlg>(bOpenFileDialog,  lpszDefExt, lpszFileName, dwFlags, lpszFilter, hWndParent)  {  // [헬프]버튼을 표시하기 위해서OFN\_SHOWHELP플래그 추가  m\_ofn.Flags |= OFN\_SHOWHELP;  }  void OnInitDone(LPOFNOTIFY lpon){  // 파일 선택 다이얼로그를 친윈도우의 중심으로 표시  GetFileDialogWindow().CenterWindow();  }  void OnHelp(LPOFNOTIFY lpon){  // 현재 열려 있는 폴더 패스를 취득  TCHAR szPath[\_MAX\_PATH];  GetFolderPath(szPath, sizeof(szPath)/sizeof(TCHAR));  CString strMsg;  strMsg.Format(\_T("현재의 폴더는 %s 입니다. "), szPath);  MessageBox(strMsg);  }  }; |

CCustomDlg 클래스의 constructor인수는 CFileDialog 클래스의 constructor인수를 참고로 하고 있습니다. constructor내에서는 파일 선택 다이얼로그에 [헬프] 버튼을 추가하기 위해서 OFN\_SHOWHELP플래그를 추가합니다. OnInitDone()에서 초기화 처리를 합니다만 여기에서는 GetFileDialogWindow()에 의해 파일 선택 다이얼로그의 윈도우 오브젝트를 취득하여 CenterWindow()에 의해 다이얼로그를 중앙에 표시하고 있습니다. OnHelp()은 [헬프] 버튼을 눌렀을 때에 불려집니다. 여기에서는 현재 열려 있는 폴더 패스를 메시지 박스로 표시하고 있을 뿐입니다.   
  
　다음에 나타내는 것은 CCustomDlg클래스를 사용하는 예입니다.

|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atldlgs.h> // common dialog를 사용하기 위해(때문에) |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_OPENDLG, OnButtonOpenDlg)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  return TRUE;  }  void OnButtonOpenDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  CCustomDlg dlg(FALSE, \_T("txt"), \_T("무제"), OFN\_OVERWRITEPROMPT,  \_T("텍스트 파일 (\*.txt)\0\*.txt\0모든 파일 (\*.\*)\0\*.\*\0\0"));  if(dlg.DoModal() == IDOK){  CString strMsg;  strMsg.Format(\_T("파일명：%s\n패스：%s"),  dlg.m\_szFileTitle, dlg.m\_szFileName);  MessageBox(strMsg);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // CommDlg.cpp  #include "stdafx.h"  #include "resource.h"  #include "CustomDlg.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치하고 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 푸쉬 버튼 | IDC\_BUTTON\_OPENDLG |

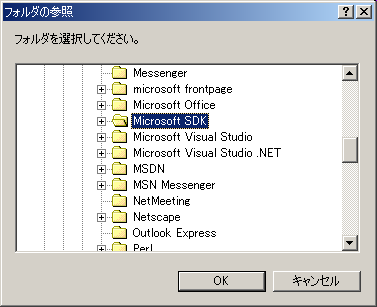
　다음에 stdafx.h 에서는 atldlgs.h 헤더를 인클루드 합니다. atldlgs.h 헤더에는 common dialog용 클래스가 정의되고 있습니다.   
  
　CMainDlg 클래스에서는 버튼용의 커멘드 메세지 핸들러 OnButtonOpenDlg()를 추가하여 거기서 CCustomDlg 클래스의 인스턴스를 작성해 DoModal()에 의해 독자적인 파일 선택 다이얼로그를 표시합니다.   
  
　마지막으로 CommDlg.cpp 에서 maindlg.h 헤더의 앞에 CustomDlg.h 헤더를 포함 합니다.

▣ 공용 다이얼로그 > 폴더 선택 다이얼로그

**< 폴더 선택 다이얼로그 >**  
  
WTL에서 폴더 선택 다이얼로그를 CFolderDialogImpl 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 이 클래스는 ATL의 CDialogImpl 클래스와 같이 base class로서의 보고 사용합니다. atldlgs.h 헤더에서는 CFileDialogImpl클래스의 상속 클래스로서 CFolderDialog클래스가 정의되고 있습니다.

|  |
| --- |
| // atldlgs.h  class CFolderDialog : public CFolderDialogImpl<CFolderDialog>  {  ...  ... |

　이하에 나타내는 것은 CFolderDialog 클래스를 사용하는 예입니다.



|  |
| --- |
| CFolderDialog dlg(NULL, \_T("폴더를 선택해 주세요. "));  if(dlg.DoModal() == IDOK){  CString strMsg;  strMsg.Format(\_T("표시명：%s\n패스：%s"),  dlg.GetFolderDisplayName(), dlg.GetFolderPath());  MessageBox(strMsg);  } |

CFolderDialog클래스의 constructor인수는 모두 생략 할 수 있습니다만 , 여기에서는 제1인수에는 부모 윈도우의 핸들, 제2인수에는 폴더 선택 다이얼로그에 표시하는 캐릭터 라인을 지정해 있습니다. 이 외 제3인수에는 플래그를 지정할 수 있습니다.   
  
폴더 선택 다이얼로그를 표시하기 위해서는 DoModal()을 사용합니다. [OK] 버튼을 눌러 폴더 선택 다이얼로그를 닫었을 경우는 반환 값으로서 IDOK가, [캔슬]을 눌러 닫았을 경우는IDCANCEL이 반환 됩니다.   
  
IDOK가 반환된 경우는 CFileDialog 클래스의 멤버인 m\_szFolderPath과 m\_szFolderDisplayName에 각각 폴더 패스와 표시명이 설정되어 있습니다.   
  
CFolderDialog 클래스의 base class인 CFolderDialogImpl클래스는 콜백 함수를 준비해 있습니다. 콜백 함수는 메세지에 의해 다음과 같은 메세지 핸들러를 호출합니다.

|  |  |
| --- | --- |
| 메세지 | 메세지 핸들러 |
| BFFM\_INITIALIZED | OnInitialized |
| BFFM\_SELCHANGED | OnSelChanged |
| BFFM\_VALIDATEFAILED | OnValidateFailed |

CFolderDialogImpl클래스의 상속 클래스는 이러한 멤버 함수를 오버라이드(override) 하는 것에 의해 폴더 선택 다이얼로그의 동작을 커스텀마이즈 할 수가 있습니다.   
  
다음에 나타내는 것은 OnInitialized()을 오버라이드(override)하여 독자적인 폴더 선택 다이얼로그를 작성하는 예입니다. 이 폴더 선택 다이얼로그는 디폴트로 현재 디렉토리를 선택하며 부모 윈도우의 중앙에 표시됩니다. 이 예에서는 CustomDlg.h이라고 하는 헤더 파일을 준비하여 거기에 CCustomDlg 클래스를 정의하고 있습니다.

|  |
| --- |
| // CustomDlg.h  class CCustomDlg : public CFolderDialogImpl<CCustomDlg>  {  public:  // constructor  CCustomDlg(HWND hWndParent = NULL, LPCTSTR lpstrTitle = NULL,  UINT uFlags = BIF\_RETURNONLYFSDIRS)  : CFolderDialogImpl<CCustomDlg>(hWndParent, lpstrTitle, uFlags)  {}  void OnInitialized(){  CWindow wnd = m\_hWnd;  wnd.CenterWindow();  TCHAR szPath[\_MAX\_PATH];  GetCurrentDirectory(sizeof(szPath)/sizeof(TCHAR), szPath);  SetSelection(szPath);  }  }; |

CCustomDlg 클래스의 constructor인수는 CFolderDialog클래스의 constructor인수를 참고로 하고 있습니다. OnInitialized() 그럼 초기화 처리를 합니다만 여기에서는 m\_hWnd으로부터 CWindow 오브젝트를 작성해 CenterWindow()에 의해 다이얼로그를 중앙에 표시하고 있습니다. 또한 현재 디렉토리의 패스를 취득해 SetSelection()에 의해 선택하고 있습니다.   
  
　다음에 나타내는 것은 CCustomDlg 클래스를 사용하는 예입니다.

|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atldlgs.h> // common dialog를 사용하기 위해 |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_OPENDLG, OnButtonOpenDlg)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  return TRUE;  }  void OnButtonOpenDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  CCustomDlg dlg(NULL, \_T("폴더를 선택해 주세요. "));  if(dlg.DoModal() == IDOK){  CString strMsg;  strMsg.Format(\_T("표시명：%s\n패스：%s"),  dlg.GetFolderDisplayName(), dlg.GetFolderPath());  MessageBox(strMsg);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // CommDlg.cpp  #include "stdafx.h"  #include "resource.h"  #include "CustomDlg.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치하여 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 푸쉬 버튼 | IDC\_BUTTON\_OPENDLG |

　다음에 stdafx.h 에서는 atldlgs.h 헤더를 인클루드 합니다.   
  
　CMainDlg클래스에서는 버튼용의 커멘드 메세지 핸들러 OnButtonOpenDlg()를 추가하여 거기서 CCustomDlg 클래스의 인스턴스를 작성해 DoModal()에 의해 독자적인 폴더 선택 다이얼로그를 표시합니다.   
  
　마지막으로 CommDlg.cpp 에서 maindlg.h 헤더의 앞에 CustomDlg.h 헤더를 인클루드 합니다.

▣ 공용 다이얼로그 > 폰트선택 다이얼로그

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **< 폴더 선택 다이얼로그 >**  　WTL에서 common dialog의 폰트 선택 다이얼로그를 CFontDialogImpl이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 이 클래스는 ATL의 CDialogImpl클래스와 같이 base class로서의 보고 사용합니다. atldlgs.h 헤더에서는 CFontDialogImpl 클래스의 상속 클래스로서 CFontDialog 클래스가 정의되고 있습니다.   |  | | --- | | // atldlgs.h  class CFontDialog : public CFontDialogImpl<CFontDialog>  {  ...  ... |   　WTL의 CFontDialog클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CFontDialog 클래스를 사용하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl14-03/01.png   |  | | --- | | CFontDialog dlg;  if(dlg.DoModal() == IDOK){  CString strMsg;  strMsg.Format(\_T("폰트명：%s\n사이즈：%dpt"),  dlg.GetFaceName(), dlg.GetSize() / 10);  MessageBox(strMsg);  } |   CFontDialog 클래스의 constructor인수는 모두 생략 할 수 있습니다. 인수는 순서로 폰트 선택 다이얼로그의 초기 상태를 설정하기 위한 LOGFONT구조체에의 포인터 , 플래그 , 프린터 디바이스 콘텍스트 , 부모 윈도우의 핸들을 지정할 수 있습니다.   폰트 선택 다이얼로그를 표시하기 위해서는 DoModal()을 사용합니다. [OK] 버튼을 눌러 폰트 선택 다이얼로그를 닫았을 경우는, 반환값으로서 IDOK가 [캔슬]을 눌러 닫았을 경우는IDCANCEL이 되돌아 옵니다.   IDOK가 반환된 경우는 CFontDialog 클래스의 멤버인 m\_cf에 폰트의 정보가 설정되어 있습니다. CFontDialog클래스는 이 정보를 취득하기 위한 멤버 함수를 준비해 있습니다.   CFontDialogImpl클래스의 상속 클래스에 메세지 맵을 준비하는 것에 의해 폰트 선택 다이얼로그의 동작을 커스텀마이즈 할 수가 있습니다.   다음에 나타내는 것은 WM\_INITDIALOG 메세지 핸들러를 준비하여 독자적인 폰트 선택 다이얼로그를 작성하는 예입니다. 이 폰트 선택 다이얼로그에는 [헬프] 버튼이 추가되어 부모 윈도우의 중앙에 표시됩니다. 이 예에서는 CustomDlg.h 이라고 하는 헤더 파일을 준비하여 거기에 CCustomDlg클래스를 정의하고 있습니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl14-03/02.png   |  | | --- | | // CustomDlg.h  class CCustomDlg : public CFontDialogImpl<CCustomDlg>  {  public:  UINT m\_uHelpMsg; // [헬프]버튼용 메세지ID  // constructor  CCustomDlg(LPLOGFONT lplfInitial = NULL,  DWORD dwFlags = CF\_EFFECTS | CF\_SCREENFONTS,  HDC hDCPrinter = NULL)  : CFontDialogImpl<CCustomDlg>(lplfInitial, dwFlags, hDCPrinter), m\_uHelpMsg(0)  {  // [헬프]버튼을 표시하기 위해서CF\_SHOWHELP플래그 추가  m\_cf.Flags |= CF\_SHOWHELP;  }  BEGIN\_MSG\_MAP\_EX(CCustomDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  MESSAGE\_HANDLER\_EX(m\_uHelpMsg, OnHelp)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  m\_cf.hwndOwner = m\_hWnd;  // [헬프]버튼용 메세지를 등록  m\_uHelpMsg = RegisterWindowMessage(HELPMSGSTRING);  CenterWindow();  return TRUE;  }  LRESULT OnHelp(UINT uMsg, WPARAM wParam, LPARAM lParam){  MessageBox(\_T("헬프 버튼을 눌렀습니다. "));  return 0;  }  }; |   CCustomDlg클래스의 constructor인수는 CFontDialog클래스의 constructor인수를 참고로 하고 있습니다. constructor내에서는 폰트 선택 다이얼로그에 [헬프] 버튼을 추가하기 위해서 CF\_SHOWHELP플래그를 추가합니다. OnInitDialog()에서 m\_cf.hwndOwner에 [헬프] 버튼용 메세지를 수신하는 윈도우 핸들 (즉 폰트 선택 다이얼로그 자신의 핸들) 을 설정해 [헬프] 버튼을 눌렀을 때의 메세지를 등록해 메세지ID를 m\_uHelpMsg로 설정합니다. 그리고 CenterWindow()에 의해 다이얼로그를 중앙에 표시하고 있습니다. OnHelp()는 [헬프] 버튼을 눌렀을 때에 불려 집니다. 여기에서는 메시지 박스로 표시하고 있을 뿐입니다.   　다음에 나타내는 것은 CCustomDlg클래스를 사용하는 예입니다.   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atldlgs.h> // common dialog를 사용하기 위해(때문에) |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_OPENDLG, OnButtonOpenDlg)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  return TRUE;  }  void OnButtonOpenDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  CCustomDlg dlg;  if(dlg.DoModal() == IDOK){  CString strMsg;  strMsg.Format(\_T("폰트명：%s\n사이즈：%dpt"),  dlg.GetFaceName(), dlg.GetSize() / 10);  MessageBox(strMsg);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // CommDlg.cpp  #include "stdafx.h"  #include "resource.h"  #include "CustomDlg.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치하여 리소스 ID를 다음과 같이 지정합니다.   |  |  | | --- | --- | | 컨트롤명 | 리소스 ID | | 푸쉬 버튼 | IDC\_BUTTON\_OPENDLG |   　다음에 stdafx.h 에서는 atldlgs.h 헤더를 인클루드 합니다.   　CMainDlg클래스에서는 버튼용의 커멘드 메세지 핸들러 OnButtonOpenDlg()를 추가하여 거기서 CCustomDlg클래스의 인스턴스를 작성해 DoModal()에 의해 독자적인 폰트 선택 다이얼로그를 표시합니다.   　마지막으로 CommDlg.cpp 에서 maindlg.h 헤더의 전에 CustomDlg.h 헤더를 인클루드 합니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 리치 에디트 폰트 선택 다이얼로그 |
| CFontDialog클래스에서는 폰트 선택 다이얼로그의 초기 상태를 설정하거나 GetCurrentFont()를 호출해 폰트 선택 다이얼로그로 선택한 항목을 취득하기 위해서LOGFONT구조체를 사용했습니다. CRichEditFontDialog클래스를 사용하면 그러한 용도를 위해서 리치 에디트 컨트롤의 서식 정보를 나타내는CHARFORMAT구조체를 사용할 수 있습니다.   　CRichEditFontDialog클래스의 base class인 CRichEditFontDialogImpl클래스는 CFontDialogImpl 클래스로부터 파생하고 있습니다. atldlgs.h 헤더에서는 다음과 같이 정의되고 있습니다.   |  | | --- | | // atldlgs.h  template <class T>  class ATL\_NO\_VTABLE CRichEditFontDialogImpl : public CFontDialogImpl< T >  {  ...  ...  };  class CRichEditFontDialog : public CRichEditFontDialogImpl<CRichEditFontDialog>  {  ...  ...  }; |   　이하에 나타내는 것은 CRichEditFontDialog클래스를 사용하는 예입니다.   |  | | --- | | // 초기 상태로10포인트로 빨강의Arial폰트를 선택  CHARFORMAT cfset = {sizeof(CHARFORMAT)};  cfset.dwMask = CFM\_FACE | CFM\_SIZE | CFM\_COLOR; // 설정하는 항목을 지정  lstrcpy(cfset.szFaceName, \_T("Arial"));  cfset.yHeight = 10 \* 20;  cfset.crTextColor = RGB(255, 0, 0);  CRichEditFontDialog dlg(cfset, CF\_SCREENFONTS | CF\_EFFECTS);  if(dlg.DoModal() == IDOK){  CHARFORMAT cfget;  dlg.GetCharFormat(cfget);  CString strMsg;  strMsg.Format(\_T("폰트명：%s\n사이즈：%dpt"),  cfget.szFaceName, cfget.yHeight / 20);  MessageBox(strMsg);  } |   CRichEditFontDialog 클래스를 사용하기 위해서는 atldlgs.h 헤더의 앞에 richedit.h헤더를 인클루드 할 필요가 있습니다. 덧붙여 atlctrls.h 헤더를 인클루드 하면 자동적으로 richedit.h 헤더도 인클루드 됩니다.   CRichEditFontDialog 클래스의 constructor의 제1인수에는 CHARFORMAT 구조체의 참조를 지정합니다. 나머지의 인수는 생략 가능하여 순서에 플래그 , 프린터 디바이스 콘텍스트 , 부모 윈도우의 핸들을 지정할 수 있습니다.   DoModal()의 호출 후 GetCharFormat()에 의해 서식 정보를 꺼냅니다. 덧붙여 CRichEditFontDialog 클래스의 base class에는 CFontDialogImpl클래스가 있기 때문에 서식 정보를 취득하기 위해서 GetFaceName()나 GetSize() 등의 멤버 함수를 사용할 수도 있습니다. | |

▣ 공용 다이얼로그 > 색 선택 다이얼로그

**< 색 선택 다이얼로그 >**  
  
WTL에서 common dialog의 색 선택 다이얼로그를 CColorDialogImpl이라고 하는 템플릿 클래스에서 캡슐화 하고 있습니다. 이 클래스는 ATL의 CDialogImpl 클래스와 같이 base class로서의 보고 사용합니다. atldlgs.h 헤더에서는 CColorDialogImpl클래스의 상속 클래스로서 CColorDialog클래스가 정의되고 있습니다.

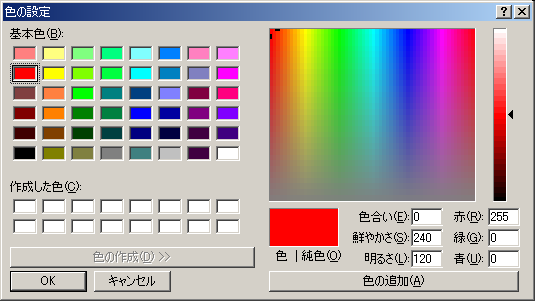
|  |
| --- |
| // atldlgs.h  class CColorDialog : public CColorDialogImpl<CColorDialog>  {  ...  ... |

　WTL의 CColorDialog 클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CColorDialog클래스를 사용하는 예입니다.



|  |
| --- |
| CColorDialog dlg(RGB(255, 0, 0));  if(dlg.DoModal() == IDOK){  CString strMsg;  strMsg.Format(\_T("R：%d\nG：%d\nB：%d"),  GetRValue(dlg.GetColor()),  GetGValue(dlg.GetColor()),  GetBValue(dlg.GetColor()));  MessageBox(strMsg);  } |

CColorDialog 클래스의 constructor 인수는 모두 생략 할 수 있습니다만, 여기에서는 제1인수에는 초기 상태로 선택되고 있는 색(빨강) 을 지정해 있습니다. 이 외 제2인수에는 플래그, 제3인수에는 부모 윈도우의 핸들을 지정할 수 있습니다.   
  
색 선택 다이얼로그를 표시하기 위해서는 DoModal()을 사용합니다. [OK] 버튼을 눌러 색 선택 다이얼로그를 닫았을 경우는, 반환 값으로서 IDOK가 [캔슬]을 눌러 닫았을 경우는IDCANCEL이 되돌아 옵니다.   
  
IDOK가 반환된 경우는 GetColor()을 호출하는 것에 의해 선택된 색을 취득할 수가 있습니다.   
  
CColorDialog 클래스의 base class인 CColorDialogImpl클래스는 메세지 맵을 준비해 있습니다. 이것은 ::RegisterWindowMessage(COLOROKSTRING)에 의해 등록된 색 선택 다이얼로그 전용의 메세지를 매핑 합니다. CColorDialogImpl클래스는 이 메세지 맵을 받으면 간접적으로 OnColorOK()라고 하는 멤버 함수를 호출합니다. OnColorOK()는 색 선택 다이얼로그의 [OK] 버튼을 눌렀을 때에 불려져 CColorDialogImpl클래스의 상속 클래스는 이OnColorOK()를 오버라이드(override) 하거나 메세지 맵을 준비하는 것에 의해 색 선택 다이얼로그의 동작을 커스텀마이즈 할 수가 있습니다.   
  
다음에 나타내는 것은 WM\_INITDIALOG 메세지 핸들러를 준비해 독자적인 색 선택 다이얼로그를 작성하는 예입니다. 이 색 선택 다이얼로그는 초기 상태로 [색의 작성] 영역을 표시해 부모 윈도우의 중앙에 표시됩니다. 그리고 [OK]버튼을 누른 후 다이얼로그를 닫았는지 어떤지의 확인 다이얼로그를 표시합니다. 이 예에서는 CustomDlg.h이라고 하는 헤더 파일을 준비하여 거기에 CustomDlg클래스를 정의하고 있습니다.



|  |
| --- |
| // CustomDlg.h  class CCustomDlg : public CColorDialogImpl<CCustomDlg>  {  public:  // constructor  CCustomDlg(COLORREF clrInit = 0, DWORD dwFlags = 0)  : CColorDialogImpl<CCustomDlg>(clrInit, dwFlags, NULL)  {  m\_cc.Flags |= CC\_FULLOPEN;  }  BEGIN\_MSG\_MAP\_EX(CCustomDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  CHAIN\_MSG\_MAP(CColorDialogImpl<CCustomDlg>)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  CenterWindow();  return TRUE;  }  BOOL OnColorOK(){  int nRet = MessageBox(\_T("색 선택 다이얼로그를 다물어도 좋습니까? "),  \_T("확인"), MB\_YESNO | MB\_ICONQUESTION);  // FALSE：다무는 ,TRUE：그대로  return (nRet == IDYES) ? FALSE : TRUE;  }  }; |

CCustomDlg 래스의 constructor인수는 CColorDialog클래스의 constructor 인수를 참고로 하고 있습니다. constructor 내에서는 초기 상태로 [색의 작성] 영역을 표시하기 위해서 CC\_FULLOPEN 플래그를 추가합니다. OnInitDialog()에서 CenterWindow()에 의해 다이얼로그를 중앙에 표시하고 있습니다. OnColorOK()은 [OK] 버튼을 눌렀을 때에 불려 집니다. 여기에서는 확인용 메시지 박스로 표시해 [네]가 선택되었을 경우에 색 선택 다이얼로그를 닫드록 하고 있습니다.   
  
　다음에 나타내는 것은 CCustomDlg클래스를 사용하는 예입니다.

|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atldlgs.h> // common dialog를 사용하기 위해(때문에) |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_OPENDLG, OnButtonOpenDlg)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  return TRUE;  }  void OnButtonOpenDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  CCustomDlg dlg(RGB(255, 0, 0));  if(dlg.DoModal() == IDOK){  CString strMsg;  strMsg.Format(\_T("R：%d\nG：%d\nB：%d"),  GetRValue(dlg.GetColor()),  GetGValue(dlg.GetColor()),  GetBValue(dlg.GetColor()));  MessageBox(strMsg);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // CommDlg.cpp  #include "stdafx.h"  #include "resource.h"  #include "CustomDlg.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 푸쉬 버튼 | IDC\_BUTTON\_OPENDLG |

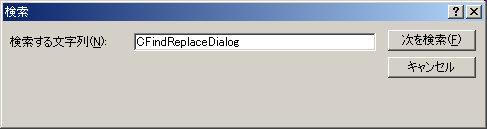
　다음에 stdafx.h 에서는 atldlgs.h 헤더를 인클루드 합니다.   
  
　CMainDlg 클래스에서는 버튼용의 커멘드 메세지 핸들러 OnButtonOpenDlg()를 추가하여 거기서 CCustomDlg 클래스의 인스턴스를 작성해 DoModal()에 의해 독자적인 색 선택 다이얼로그를 표시합니다.   
  
　마지막으로 CommDlg.cpp 안에서 maindlg.h 헤더의 앞에 CustomDlg.h 헤더를 인클루드 합니다.

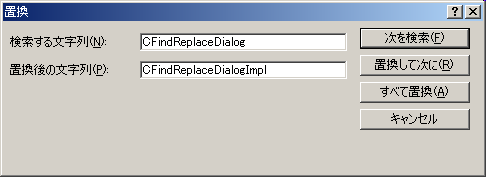
▣ 공용 다이얼로그 > 검색 치환 선택 다이얼로그

**< 검색 치환 다이얼로그** >　  
  
WTL에서 common dialog의 검색 다이얼로그와 치환 다이얼로그를 CFindReplaceDialogImpl이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 이 클래스는 ATL의 CDialogImpl클래스와 같이 base class로서의 보고 사용합니다. atldlgs.h 헤더에서는 CFindReplaceDialogImpl클래스의 상속 클래스로서 CFindReplaceDialog 클래스가 정의되고 있습니다.

|  |
| --- |
| // atldlgs.h  class CFindReplaceDialog : public CFindReplaceDialogImpl<CFindReplaceDialog>  {  ...  ... |

　WTL의 CFindReplaceDialog클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 CFindReplaceDialog 클래스를 사용해 검색 다이얼로그와 치환 다이얼로그를 작성하는 예입니다. 다이얼로그상의 에디트 컨트롤에 입력된 캐릭터 라인에 대해 검색이나 치환을 실행합니다.





|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atldlgs.h> // common dialog를 사용하기 위해(때문에) |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  enum { MAX\_LINE = 256};  CEdit m\_edit\_line;  bool m\_bOpen;  // constructor  CMainDlg() : m\_bOpen(false)  {}  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MESSAGE\_HANDLER\_EX(CFindReplaceDialog::GetFindReplaceMsg(), OnFindReplace)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_FINDDLG, OnButtonFindDlg)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_REPLACEDLG, OnButtonReplaceDlg)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_edit\_line = GetDlgItem(IDC\_EDIT\_LINE);  m\_edit\_line.LimitText(MAX\_LINE - 1);  return TRUE;  }  void OnButtonFindDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  OpenFindReplaceDlg(TRUE);  }  void OnButtonReplaceDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  OpenFindReplaceDlg(FALSE);  }  void OpenFindReplaceDlg(BOOL bFind){  if(m\_bOpen)  return;  m\_bOpen = true;  CFindReplaceDialog\* pDlg = new CFindReplaceDialog;  pDlg->Create(bFind, NULL, NULL,  FR\_HIDEWHOLEWORD | FR\_HIDEMATCHCASE | FR\_HIDEUPDOWN);  pDlg->ShowWindow(SW\_SHOW);  }  LRESULT OnFindReplace(UINT uMsg, WPARAM wParam, LPARAM lParam){  CFindReplaceDialog\* pDlg = CFindReplaceDialog::GetNotifier(lParam);  // 다이얼로그가 닫혀졌을 경우  if(pDlg->IsTerminating()){  m\_bOpen = false;  return 0;  }  TCHAR szLine[MAX\_LINE];  m\_edit\_line.GetWindowText(szLine, sizeof(szLine)/sizeof(TCHAR));  const CString strLine = szLine;  const CString strFind = pDlg->GetFindString();  const CString strReplace = pDlg->GetReplaceString();  if(pDlg->FindNext()){  int nStartPos, nEndPos;  nStartPos = nEndPos = 0;  m\_edit\_line.GetSel(nStartPos, nEndPos);  int nFindPos = strLine.Find(strFind, nEndPos);  if(nFindPos != -1)  m\_edit\_line.SetSel(nFindPos, nFindPos + strFind.GetLength());  }else if(pDlg->ReplaceCurrent()){  int nStartPos, nEndPos;  nStartPos = nEndPos = 0;  m\_edit\_line.GetSel(nStartPos, nEndPos);  int nGap = 0;  if((nEndPos - nStartPos) > 0){  m\_edit\_line.ReplaceSel(strReplace);  nGap = strReplace.GetLength() - strFind.GetLength();  }  int nFindPos = strLine.Find(strFind, nEndPos);  if(nFindPos != -1){  m\_edit\_line.SetSel(nFindPos + nGap,  nFindPos + strFind.GetLength() + nGap);  }  }else if(pDlg->ReplaceAll()){  int nFindPos = 0;  int nFindCount = 0;  const int nGap = strReplace.GetLength() - strFind.GetLength();  while((nFindPos = strLine.Find(strFind, nFindPos)) != -1){  m\_edit\_line.SetSel(nFindPos + nGap \* nFindCount,  nFindPos + strFind.GetLength() + nGap \* nFindCount);  m\_edit\_line.ReplaceSel(strReplace);  nFindCount++;  nFindPos += strFind.GetLength();  }  }  return 0;  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // CommDlg.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤과 에디트 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 에디트 컨트롤의 [스타일]에서는 디폴트에 가세해 [항상 선택을 표시]에 체크를 넣습니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 푸쉬 버튼 | IDC\_BUTTON\_FINDDLG |
| 푸쉬 버튼 | IDC\_BUTTON\_REPLACEDLG |
| 에디트 | IDC\_EDIT\_LINE |

　다음에 stdafx.h 에서는 atlctrls.h 헤더와 atldlgs.h 헤더를 인클루드 합니다.   
  
　CMainDlg클래스에서는 우선 enum에 의해 에디트 컨트롤의 최대 문자수를 MAX\_LINE이라고 정의합니다. 牡슴?/span> 에디트 컨트롤용으로 CEdit 클래스의 인스턴스를 멤버 변수로서 선언합니다. 이것들을 사용하기 위해서는 WM\_INITDIALOG 메세지 핸들러로 컨트롤의 핸들을 대입할 필요가 있습니다. 그리고 검색 또는 치환 다이얼로그가 현재 열려 있는지 어떤지를 나타내는 플래그 m\_bOpen를 선언합니다. 이것은 constructor에 의해 false 초기화합니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러로 LimitText()에 의해 에디트 컨트롤의 최대 문자수를 설정합니다.   
  
　다음에 2개의 버튼 컨트롤용 커멘드 메세지 핸들러를 추가합니다. 이러한 메세지 핸들러는 양쪽 모두 OpenFindReplaceDlg()라고 하는 독자적인 함수를 호출합니다. OpenFindReplaceDlg()는 검색 또는 치환 다이얼로그를 여는 함수로 인수에BOOL값을 받아 TRUE라면 검색 다이얼로그를 FALSE 이라면 치환 다이얼로그를 엽니다. 덧붙여 벌써 어느 쪽인가의 다이얼로그가 열려 있는 경우는 굳이 하지 않고 리턴 합니다.   
  
　검색 또는 치환 다이얼로그는 모달리스 다이얼로그입니다. 이 때문에 CFindReplaceDialog 클래스의 오브젝트는 new에 의해 heap상에 작성할 필요가 있습니다. (그렇게 하지 않으면 OpenFindReplaceDlg() 함수의 스코프를 빠졌을 때에 CFindReplaceDialog클래스의 오브젝트도 파기되어 버립니다. ) 검색 또는 치환 다이얼로그를 작성하기 위해서는 Create()을 호출합니다. 제1인수는 BOOL 값으로 TRUE라면 검색 다이얼로그를 FALSE 이라면 치환 다이얼로그를 작성합니다. 제2인수에는 검색하는 캐릭터 라인 제3인수에는 치환 후의 캐릭터 라인 , 제4인수에는 플래그 , 제5인수에는 부모 윈도우의 핸들을 지정할 수 있어 제3인수 이후는 생략 가능합니다. 덧붙여 이번 예에서는 예를 단순화 하기 위한 플래그에 FR\_HIDEWHOLEWORD, FR\_HIDEMATCHCASE, FR\_HIDEUPDOWN를 지정했습니다.   
  
　다음에 메세지 맵에 다음과 같은 엔트리를 추가합니다.

|  |
| --- |
| BEGIN\_MSG\_MAP\_EX(CMainDlg)  MESSAGE\_HANDLER\_EX(CFindReplaceDialog::GetFindReplaceMsg(), OnFindReplace)  ...  ... |

CFindReplaceDialog::GetFindReplaceMsg()는 검색 또는 치환 다이얼로그용의 메세지를 등록해 그 메시지 ID를 돌려줍니다. 메세지 맵에 의해 이 메시지 ID와 OnFindReplace()를 묶는 것에 의해 검색 또는 치환 다이얼로그로부터 메세지가 발생하면 OnFindReplace()가 불려 가게 됩니다. 다음에 나타내는 것은 메세지가 발생하는 타이밍이라고 메세지의 종류를 특정하는 멤버 함수의 대응표입니다.

|  |  |
| --- | --- |
| 타이밍 | 멤버 함수명 |
| [다음을 검색]버튼을 눌렀을 때 | FindNext |
| [치환해 다음에]버튼을 눌렀을 때 | ReplaceCurrent |
| [모두 치환]버튼을 눌렀을 때 | ReplaceAll |
| 검색 또는 치환 다이얼로그를 다물었을 때 | IsTerminating |

OnFindReplace() 동안으로는 이러한 멤버 함수를 호출하는 것에 의해 어느 타이밍에 OnFindReplace()가 불려 갔는지를 판정해 각각의 타이밍에 응한 처리를 실시하고 있습니다.   
  
　그런데 위의 예에서는 CFindReplaceDialog 클래스의 오브젝트는 new에 의해 작성합니다만 delete에 의해 오브젝트를 삭제하고 있는 코드가 눈에 띄지 않습니다. 이것은 CFindReplaceDialog 클래스의 오브젝트는 검색 또는 치환 다이얼로그를 닫았을 때에 스스로를 삭제하기 때문입니다.   
  
CFindReplaceDialogImpl클래스의 상속 클래스에 메세지 맵을 준비하는 것에 의해 검색 치환 다이얼로그의 동작을 커스텀마이즈 할 수가 있습니다.   
  
다음에 나타내는 것은 WM\_INITDIALOG 메세지 핸들러와 GetFindReplaceMsg()에 의한 검색 또는 치환 다이얼로그 전용 메세지 핸들러를 준비해 검색 또는 치환 처리를 캡슐화하는 독자적인 검색 치환 다이얼로그를 작성하는 예입니다. 이 검색 치환 다이얼로그는 부모 윈도우의 중앙에 표시됩니다. 이 예에서는 CustomDlg.h이라고 하는 헤더 파일을 준비해 거기에 CCustomDlg 클래스를 정의하고 있습니다.

|  |
| --- |
| // CustomDlg.h  class CCustomDlg : public CFindReplaceDialogImpl<CCustomDlg>  {  static bool m\_bOpenDlg;  CEdit& m\_edit;  public:  // constructor  CCustomDlg(CEdit& edit) : m\_edit(edit){  m\_fr.Flags = m\_fr.Flags | FR\_HIDEWHOLEWORD | FR\_HIDEMATCHCASE | FR\_HIDEUPDOWN;  }  BEGIN\_MSG\_MAP\_EX(CCustomDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  MESSAGE\_HANDLER\_EX(GetFindReplaceMsg(), OnFindReplace)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  CenterWindow();  m\_bOpenDlg = true;  m\_fr.hwndOwner = m\_hWnd;  return TRUE;  }  LRESULT OnFindReplace(UINT uMsg, WPARAM wParam, LPARAM lParam){  // 다이얼로그가 닫혀졌을 경우  if(IsTerminating()){  m\_bOpenDlg = false;  return 0;  }  TCHAR\* pLine = new TCHAR[m\_edit.LineLength() + 1];  m\_edit.GetWindowText(pLine, m\_edit.LineLength() + 1);  const CString strLine = pLine;  delete [] pLine;  const CString strFind = GetFindString();  const CString strReplace = GetReplaceString();  if(FindNext()){  int nStartPos, nEndPos;  nStartPos = nEndPos = 0;  m\_edit.GetSel(nStartPos, nEndPos);  int nFindPos = strLine.Find(strFind, nEndPos);  if(nFindPos != -1)  m\_edit.SetSel(nFindPos, nFindPos + strFind.GetLength());  }else if(ReplaceCurrent()){  int nStartPos, nEndPos;  nStartPos = nEndPos = 0;  m\_edit.GetSel(nStartPos, nEndPos);  int nGap = 0;  if((nEndPos - nStartPos) > 0){  m\_edit.ReplaceSel(strReplace);  nGap = strReplace.GetLength() - strFind.GetLength();  }  int nFindPos = strLine.Find(strFind, nEndPos);  if(nFindPos != -1)  m\_edit.SetSel(nFindPos + nGap, nFindPos + strFind.GetLength() + nGap);  }else if(ReplaceAll()){  int nFindPos = 0;  int nFindCount = 0;  const int nGap = strReplace.GetLength() - strFind.GetLength();  while((nFindPos = strLine.Find(strFind, nFindPos)) != -1){  m\_edit.SetSel(nFindPos + nGap \* nFindCount,  nFindPos + strFind.GetLength() + nGap \* nFindCount);  m\_edit.ReplaceSel(strReplace);  nFindCount++;  nFindPos += strFind.GetLength();  }  }  return 0;  }  static bool IsOpen(){  return m\_bOpenDlg;  }  };  bool CCustomDlg::m\_bOpenDlg = false; |

CCustomDlg클래스의 constructor의 인수에는 CEdit오브젝트의 참조를 지정합니다. CCustomDlg클래스는 여기서 지정한 에디트 컨트롤에 대해서 검색이나 치환 처리를 실시합니다. OnInitDialog()에서 m\_bOpenDlg이라고 하는 정적 멤버 변수에 true를 대입합니다. 이 변수는 검색 또는 치환 다이얼로그가 열려 있는지 어떤지를 나타내는 플래그입니다. 외부로부터IsOpen()를 호출하는 것에 의해 이 플래그의 값을 취득할 수 있습니다. 그리고 OnInitDialog()에서 m\_fr.hwndOwner에 자신의 핸들을 대입하고 있습니다. 이것에 의해 검색 또는 치환 다이얼로그 전용의 메세지가 자기 자신에게 보내져 옵니다. 메세지가 보내져 오면 OnFindReplace()가 불려 집니다. 이 함수에서는 constructor인수로 지정된 에디트 컨트롤에 대해서 검색이나 치환 처리를 실시합니다.   
  
　다음에 나타내는 것은 전술의 CFindReplaceDialog 클래스를 사용한 프로그램을 CCustomDlg 클래스를 사용해 고쳐 쓰는 예입니다.

|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atldlgs.h> // common dialog를 사용하기 위해(때문에) |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  enum { MAX\_LINE = 256};  CEdit m\_edit\_line;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_FINDDLG, OnButtonFindDlg)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_REPLACEDLG, OnButtonReplaceDlg)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_edit\_line = GetDlgItem(IDC\_EDIT\_LINE);  m\_edit\_line.LimitText(MAX\_LINE - 1);  return TRUE;  }  void OnButtonFindDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  OpenFindReplaceDlg(TRUE);  }  void OnButtonReplaceDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  OpenFindReplaceDlg(FALSE);  }  void OpenFindReplaceDlg(BOOL bFind){  if(CCustomDlg::IsOpen())  return;  CCustomDlg\* pDlg = new CCustomDlg(m\_edit\_line);  pDlg->Create(bFind, NULL);  pDlg->ShowWindow(SW\_SHOW);  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // CommDlg.cpp  #include "stdafx.h"  #include "resource.h"  #include "CustomDlg.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

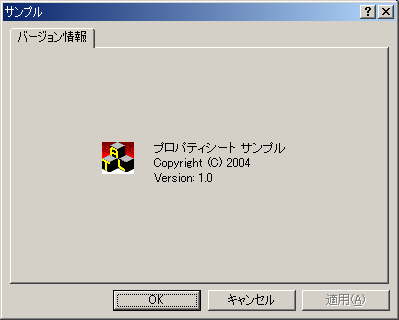
　OpenFindReplaceDlg() 안에서 독자적인 검색 치환 다이얼로그를 엽니다. 우선 CCustomDlg::IsOpen()에 의해 벌써 검색 또는 치환 다이얼로그가 열려 있는지 어떤지 체크해 열려 있지 않으면 CCustomDlg 클래스의 오브젝트를 작성합니다.   
  
　마지막으로 CommDlg.cpp 안에서 maindlg.h 헤더의 앞에 CustomDlg.h 헤더를 인클루드 합니다.

▣ 프로퍼티 쉬트 > 프로퍼티 쉬트

**< 프롭퍼티 시트 >**  
  
WTL에서 프롭퍼티 시트와 그 프롭퍼티 시트 상에 표시하는 탭(프롭퍼티 페이지)을 각각 CPropertySheetImpl, CPropertyPageImpl 라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 이러한 클래스는 ATL의 CDialogImpl클래스와 같이 base class로서의 보고 사용합니다. atldlgs.h 헤더에서는 각각의 상속 클래스로서 CPropertySheet클래스와 CPropertyPage클래스가 정의되고 있습니다.

|  |
| --- |
| // atldlgs.h  class CPropertySheet : public CPropertySheetImpl<CPropertySheet>  {  ...  ...  template <WORD t\_wDlgTemplateID>  class CPropertyPage : public CPropertyPageImpl<CPropertyPage<t\_wDlgTemplateID> >  {  ...  ... |

　WTL의 CPropertySheet 클래스와 CPropertyPage클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 아래에 나타내는 것은 CPropertySheet클래스와 CPropertyPage 클래스를 사용하는 간단한 예 입니다. 다이얼로그 자원을 프롭퍼티 페이지에 관련 지어 그 페이지를 프롭퍼티 시트에 추가해 표시할 뿐입니다. 덧붙여 WTL의 프롭퍼티 시트 관련 클래스를 사용하기 위해서는 atldlgs.h 헤더를 인클루드 할 필요가 있습니다.



|  |
| --- |
| CPropertyPage<IDD\_PROPPAGE\_VERSION> page;  CPropertySheet sheet(\_T("샘플"));  sheet.AddPage(page);  if(sheet.DoModal() == IDOK){  // [OK]버튼을 눌러 닫았을 때의 처리  } |

우선 프로젝트에 프롭퍼티 페이지용의 다이얼로그 자원(IDD\_PROPPAGE\_MEDIUM)을 추가해 ID를IDD\_PROPPAGE\_VERSION로 변경합니다. 그리고 그 다이얼로그 자원에 스태틱 컨트롤을 배치해 캐릭터 라인이나 아이콘을 설정합니다. 이 다이얼로그 자원을 CPropertyPage 클래스의 템플릿 인수로 지정하는 것으로 프롭퍼티 페이지에 다이얼로그 자원을 관련 지을 수 있습니다. 프롭퍼티 페이지의 탭 부분에는 다이얼로그 자원의 타이틀 캐릭터 라인이 표시됩니다. 덧붙여 CPropertyPage 클래스의 constructor에 캐릭터 라인을 지정하면 그 캐릭터 라인이 탭 부분의 캐릭터 라인으로서 설정됩니다.   
  
다음에 CPropertySheet 클래스의 인스턴스를 작성해 AddPage()에 의해 프롭퍼티 페이지를 프롭퍼티 시트에 추가합니다. 프롭퍼티 시트에는 AddPage()을 여러 차례 호출하는 것에 의해 복수의 프롭퍼티 페이지를 추가할 수 있습니다. CPropertySheet 클래스의 경우도 constructor에 캐릭터 라인을 지정하는 것에 의해 프롭퍼티 시트의 타이틀 캐릭터 라인을 설정할 수가 있습니다.   
  
프롭퍼티 시트를 모달로 표시하기 위해서는 DoModal()을 사용합니다. [OK] 버튼을 눌러 파일 선택 다이얼로그를 닫은 경우는 반환값으로서 IDOK가 [캔슬]을 눌러 닫았을 경우는IDCANCEL이 반환됩니다. 덧붙여 Create()를 사용하면 프롭퍼티 시트를 모달리스로 표시할 수 있습니다.   
  
　위의 예 에서는 단지 다이얼로그 자원상에 스태틱 컨트롤을 배치해 캐릭터 라인이나 아이콘 표시하는 것만으로 했다. 그러나 프롭퍼티 시트의 실제의 용도는 다이얼로그 리소스 상의 여러가지 컨트롤에 데이터를 설정하거나 그 데이터를 외부의 클래스와 교환합니다. 그런데, 다음은 CPropertySheetImpl클래스나 CPropertyPageImpl클래스로부터 상속 클래스를 작성하여 프롭퍼티 시트의 동작을 커스텀마이즈 하는 예를 나타냅니다.



우선 프로젝트에 프롭퍼티 페이지용의 다이얼로그 리소스(IDD\_PROPPAGE\_MEDIUM)를 추가해 ID를 IDD\_PROPPAGE\_WINDOW로 변경합니다. 이 다이얼로그 리소스에는 ID가 IDC\_CHECK\_TOPMOST의 체크 박스를 추가합니다.   
  
다음에 CustomSheet.h 라고 하는 헤더 파일을 준비하여 거기에 CCustomPage 라고 하는 프롭퍼티 페이지 클래스를 정의합니다.

|  |
| --- |
| // CustomSheet.h  class CCustomPage : public CPropertyPageImpl<CCustomPage>  {  CButton m\_button\_topmost;  public:  enum { IDD = IDD\_PROPPAGE\_WINDOW };  bool m\_bTopmost;  // constructor  CCustomPage(ATL::\_U\_STRINGorID title = (LPCTSTR)NULL)  : CPropertyPageImpl<CCustomPage>(title)  {}  BEGIN\_MSG\_MAP\_EX(CCustomPage)  MSG\_WM\_INITDIALOG(OnInitDialog)  CHAIN\_MSG\_MAP(CPropertyPageImpl<CCustomPage>)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  m\_button\_topmost = GetDlgItem(IDC\_CHECK\_TOPMOST);  m\_button\_topmost.SetCheck(m\_bTopmost);  return TRUE;  }  BOOL OnKillActive(){  m\_bTopmost = (m\_button\_topmost.GetCheck() != 0);  return TRUE;  }  }; |

CCustomPage클래스는 CPropertyPageImpl클래스로부터 파생하고 있어(템플릿 인수는 아니고 ) enum에 의해 직접 다이얼로그 리소스 (IDD\_PROPPAGE\_WINDOW)을 관련 짓고 있습니다. 그리고, 체크 박스의 체크 상태를 나타내는 m\_bTopmost라고 하는 bool 형태 멤버 변수를 가지고 있어 이 변수를 사용해 외부의 클래스와 데이터의 교환을 합니다. CCustomPage 클래스의 WM\_INITDIALOG 메세지 핸들러에서는 이 변수의 값으로 체크 박스 상태를 초기화해 OnKillActive() 안에서 체크 박스 상태를 m\_bTopmost에 보존합니다. OnKillActive()는 프롭퍼티 페이지가 포커스를 잃을 때([OK] 버튼을 눌러 프롭퍼티 시트를 닫을 때도 포함한다)에 불려 집니다. 즉, 프롭퍼티 시트(프롭퍼티 페이지)를 표시하기 전에 m\_bTopmost에 값을 설정하여 프롭퍼티 시트(프롭퍼티 페이지)를 닫은 후에 m\_bTopmost의 값을 취득하는 것에 의해 프롭퍼티 시트와 외부의 윈도우가 체크 박스 상태를 공유할 수가 있습니다.   
  
　덧붙여 CCustomPage 클래스의 메세지 맵에서는 base class인 CPropertyPageImpl클래스에의 체인을 추가하고 있습니다. CPropertyPageImpl클래스는 메세지 맵을 준비하고 있어 이것은 프롭퍼티 페이지에 대한 WM\_NOTIFY 메세지를 매핑 합니다. WM\_NOTIFY 메세지 핸들러에서는 통지 코드에 의해 다음과 같은 멤버 함수를 호출합니다.

|  |  |
| --- | --- |
| 통지 코드 | 멤버 함수명 |
| PSN\_SETACTIVE | OnSetActive |
| PSN\_KILLACTIVE | OnKillActive |
| PSN\_APPLY | OnApply |
| PSN\_RESET | OnReset |
| PSN\_QUERYCANCEL | OnQueryCancel |
| PSN\_WIZNEXT | OnWizardNext |
| PSN\_WIZBACK | OnWizardBack |
| PSN\_WIZFINISH | OnWizardFinish |
| PSN\_HELP | OnHelp |
| PSN\_GETOBJECT | OnGetObject |
| PSN\_TRANSLATEACCELERATOR | OnTranslateAccelerator |
| PSN\_QUERYINITIALFOCUS | OnQueryInitialFocus |

CPropertyPageImpl 클래스의 상속 클래스는 이러한 멤버 함수를 오버라이드(override)하는 것에 의해 프롭퍼티 페이지의 동작을 커스텀마이즈 할 수가 있습니다.   
  
　다음과 같이 CustomSheet.h 파일 내에 CCustomSheet 라고 하는 프롭퍼티 시트 클래스를 정의합니다.

|  |
| --- |
| // CustomSheet.h  class CCustomPage : public CPropertyPageImpl<CCustomPage>  {  ...  };  class CCustomSheet : public CPropertySheetImpl<CCustomSheet>  {  public:  CCustomPage m\_pageCustom;  CPropertyPage<IDD\_PROPPAGE\_VERSION> m\_pageVersion;  // constructor  CCustomSheet(ATL::\_U\_STRINGorID title = (LPCTSTR)NULL,  UINT uStartPage = 0, HWND hWndParent = NULL)  : CPropertySheetImpl<CCustomSheet>(title, uStartPage, hWndParent)  {  m\_psh.dwFlags |= PSH\_NOAPPLYNOW;  AddPage(m\_pageCustom);  AddPage(m\_pageVersion);  }  BEGIN\_MSG\_MAP\_EX(CCustomSheet)  CHAIN\_MSG\_MAP(CPropertySheetImpl<CCustomSheet>)  END\_MSG\_MAP()  static int CALLBACK PropSheetCallback(HWND hWnd, UINT uMsg, LPARAM lParam){  if(uMsg == PSCB\_PRECREATE){  // 문맥 헬프 버튼을 삭제  ((LPDLGTEMPLATE)lParam)->style &= ~DS\_CONTEXTHELP;  return 0;  }  return CPropertySheetImpl<CCustomSheet>::PropSheetCallback(hWnd, uMsg, lParam);  }  }; |

CCustomSheet클래스의 constructor인수는 CPropertySheet 클래스의 constructor 인수를 참고로 하고 있습니다. CCustomSheet 클래스의 constructor에서는 프롭퍼티 시트의 [적용] 버튼을 비 표시로 하기 위해서 PSH\_NOAPPLYNOW 플래그를 추가하고 있습니다. 그리고, 멤버 변수로서 선언한 2개의 프롭퍼티 페이지를AddPage()에 의해 추가하고 있습니다. CCustomSheet 클래스에서는 프롭퍼티 시트의 문맥 헬프 버튼( 타이틀 바 위의[?]버튼) 을 비표시로 하기 위해서 PropSheetCallback() 콜백 함수를 오버라이드(override) 하고 있습니다. 이 함수에서는 PSCB\_PRECREATE 메세지가 보내져 왔을 때에 DS\_CONTEXTHELP스타일을 삭제하고 있습니다. 덧붙여 시스템에 IE5 이상이 인스톨 되고 있는 경우는 constructor으로 PSH\_NOCONTEXTHELP 플래그를 추가하는 것으로 문맥 헬프 버튼을 비 표시로 할 수 있습니다.   
  
　다음에 나타내는 것은 CCustomSheet 클래스를 사용하는 예입니다.

|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atldlgs.h> // common dialog를 사용하기 위해(때문에) |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  bool m\_bTopmost;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_OPENDLG, OnButtonOpenDlg)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  m\_bTopmost = false;  return TRUE;  }  void OnButtonOpenDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  CCustomSheet sheet(\_T("샘플"));  sheet.m\_pageCustom.m\_bTopmost = m\_bTopmost;  if(sheet.DoModal() == IDOK){  m\_bTopmost = sheet.m\_pageCustom.m\_bTopmost;  SetWindowPos(m\_bTopmost ? HWND\_TOPMOST : HWND\_NOTOPMOST,  0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE | SWP\_SHOWWINDOW);  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // CommDlg.cpp  #include "stdafx.h"  #include "resource.h"  #include "CustomSheet.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 푸쉬 버튼 | IDC\_BUTTON\_OPENDLG |

　다음에 stdafx.h 에서는 atldlgs.h 헤더를 인클루드 합니다.   
  
　CMainDlg 클래스에서는 버튼용의 커멘드 메세지 핸들러 OnButtonOpenDlg()를 추가해 거기서 CCustomSheet 클래스의 인스턴스를 작성해 DoModal()에 의해 독자적인 프롭퍼티 시트를 표시합니다.   
  
　마지막으로 CommDlg.cpp 에서 maindlg.h 헤더의 앞에 CustomSheet.h 헤더를 인클루드 합니다.

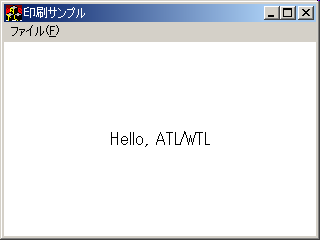
▣ 프로퍼티 쉬트 > 위자드 모드

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **< 위저드 모드 >**  전회는 통상의 프롭퍼티 시트를 작성했습니다만 CPropertySheetImpl클래스의 멤버 함수인 SetWizardMode() 을 호출하는 것에 의해 위저드 타입의 프롭퍼티 시트를 작성할 수가 있습니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl15-02/01.png   |  | | --- | | CPropertyPage<IDD\_PROPPAGE\_VERSION> page;  CPropertySheet sheet;  sheet.AddPage(page);  sheet.SetWizardMode(); // 위저드 모드로 설정  sheet.DoModal(); |   우선 전회 작성한 프롭퍼티 페이지용의 다이얼로그 자원 IDD\_PROPPAGE\_VERSION 을 지정해 프롭퍼티 페이지를 작성해 그것을 프롭퍼티 시트에 추가합니다. 그리고 SetWizardMode()에 의해 위저드 모드로 합니다. 그리고 다이얼로그 자원의 타이틀 캐릭터 라인이 위저드의 타이틀로서 표시됩니다. 이와 같이 위저드 모드의 프롭퍼티 시트의 작성은 통상의 프롭퍼티 시트의 경우와 서로 너무 바뀝니다.   　그런데 위의 예에서는 [완료]버튼이 없습니다. 그리고 프롭퍼티 페이지상의 컨트롤이 있는 조건을 채웠을 경우만 [다음에] 버튼을 유효하게 하고 싶다고 하는 일이 있겠지요. 그러한 경우는 CPropertyPageImpl클래스로부터 파생시킨 독자적인 자산 페이지 클래스를 작성해 그 중에SetWizardButtons()를 호출해 버튼 상태를 설정합니다.   　다음에 나타내는 것은 CPropertySheetImpl 클래스나 CPropertyPageImpl클래스로부터 상속 클래스를 작성하여 독자적인 위저드 모드 프롭퍼티 시트를 작성하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl15-02/02.png  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl15-02/03.png  우선 프로젝트에 프롭퍼티 페이지용의 다이얼로그 자원을 2개 추가하여 각각의 ID를 IDD\_PROPPAGE\_CUSTOM과 IDD\_PROPPAGE\_FINISH로 변경합니다. 1번째의 다이얼로그 리소스(ID:IDD\_PROPPAGE\_CUSTOM)에는 ID가 IDC\_CHECK\_NEXT 의 체크 박스를 추가합니다.   다음으로 CustomWizard.h라고 하는 헤더 파일을 준비해 거기에 CCustomPage, CFinishPage라고 하는 프롭퍼티 페이지 클래스와 CCustomWizard라고 하는 프롭퍼티 시트 클래스를 정의합니다.   |  | | --- | | // CustomWizard.h  class CCustomPage : public CPropertyPageImpl<CCustomPage>  {  CButton m\_button\_topmost;  public:  enum { IDD = IDD\_PROPPAGE\_CUSTOM };  // constructor  CCustomPage(ATL::\_U\_STRINGorID title = (LPCTSTR)NULL)  : CPropertyPageImpl<CCustomPage>(title)  {}  BEGIN\_MSG\_MAP\_EX(CCustomPage)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_CHECK\_NEXT, OnCheckNext)  CHAIN\_MSG\_MAP(CPropertyPageImpl<CCustomPage>)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  m\_button\_topmost = GetDlgItem(IDC\_CHECK\_NEXT);  return TRUE;  }  void OnCheckNext(UINT uNotifyCode, int nID, HWND hWndCtl){  OnSetActive();  }  BOOL OnSetActive(){  // 체크 박스가 체크되고 있는 경우는[다음에]버튼을 유효하게 해 ,  // 그렇지 않으면 무효로 한다. [돌아오는]버튼은 항상 무효.  SetWizardButtons(m\_button\_topmost.GetCheck() ? PSWIZB\_NEXT : 0);  return TRUE;  }  };  class CFinishPage : public CPropertyPageImpl<CFinishPage>  {  public:  enum { IDD = IDD\_PROPPAGE\_FINISH };  // constructor  CFinishPage(ATL::\_U\_STRINGorID title = (LPCTSTR)NULL)  : CPropertyPageImpl<CFinishPage>(title)  {}  BEGIN\_MSG\_MAP\_EX(CFinishPage)  CHAIN\_MSG\_MAP(CPropertyPageImpl<CFinishPage>)  END\_MSG\_MAP()  BOOL OnSetActive(){  // [돌아오는]버튼과[완료]버튼을 유효하게 한다.  // 덧붙여[다음에]버튼은[완료]버튼으로 옮겨진다.  SetWizardButtons(PSWIZB\_BACK | PSWIZB\_FINISH);  return TRUE;  }  };  class CCustomWizard : public CPropertySheetImpl<CCustomWizard>  {  public:  CCustomPage m\_pageCustom;  CFinishPage m\_pageFinish;  // constructor  CCustomWizard(ATL::\_U\_STRINGorID title = (LPCTSTR)NULL,  UINT uStartPage = 0, HWND hWndParent = NULL)  : CPropertySheetImpl<CCustomWizard>(title, uStartPage, hWndParent)  {  SetWizardMode(); // 위저드 모드로 설정  AddPage(m\_pageCustom);  AddPage(m\_pageFinish);  }  BEGIN\_MSG\_MAP\_EX(CCustomWizard)  CHAIN\_MSG\_MAP(CPropertySheetImpl<CCustomWizard>)  END\_MSG\_MAP()  }; |   CCustomPage 클래스는 위저드의 1페이지째를 나타냅니다. 이 프롭퍼티 페이지에는 체크 박스가 한 개 있어 체크를 넣으면 SetWizardButtons(PSWIZB\_NEXT)가 불려져 [다음에] 버튼이 유효하게 됩니다. 덧붙여OnSetActive()는 프롭퍼티 페이지가 액티브하게 될 때마다 불려 갑니다만 프롭퍼티 페이지가 최초로 표시되었을 때는 체크 박스는 체크되어 있지 않기 때문에 SetWizardButtons(0)가 불려져 [돌아오는] 버튼도 [다음에] 버튼도 무효가 됩니다.   CFinishPage클래스는 위저드의 2페이지째를 나타냅니다. 이 프롭퍼티 페이지는 스태틱 컨트롤로 메세지를 표시하고 있을 뿐입니다. 덧붙여 이 페이지가 위저드의 최후이므로OnSetActive()에서는 PSWIZB\_FINISH을 지정해 SetWizardButtons()를 호출하는 것으로 [다음에] 버튼을[완료]버튼으로 변경하고 있습니다. 이와 같이 [다음에] 버튼과 [완료] 버튼은 공통의 버튼을 사용합니다.   CCustomWizard 클래스는 상기의 2개의 프롭퍼티 페이지 클래스의 인스턴스를 멤버 변수로 가져 constructor 　 으로 AddPage()를 호출하는 것에 의해 추가하고 있습니다. 그리고 constructor에서는 SetWizardMode()을 호출해 프롭퍼티 시트를 위저드 모드로 하고 있습니다.   　다음에 나타내는 것은 CCustomWizard 클래스를 사용하는 예입니다.   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atldlgs.h> // common dialog를 사용하기 위해(때문에) |  |  | | --- | | // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_OPENDLG, OnButtonOpenDlg)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  return TRUE;  }  void OnButtonOpenDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  CCustomWizard wizard;  if(wizard.DoModal() == IDOK){  // [완료]버튼을 눌러 닫았을 때의 처리  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |  |  | | --- | | // CommDlg.cpp  #include "stdafx.h"  #include "resource.h"  #include "CustomWizard.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다.   |  |  | | --- | --- | | 컨트롤명 | 리소스 ID | | 푸쉬 버튼 | IDC\_BUTTON\_OPENDLG |   　다음에 stdafx.h 에서는 atldlgs.h 헤더를 인클루드 합니다.  　CMainDlg클래스에서는 버튼용의 커멘드 메세지 핸들러 OnButtonOpenDlg()를 추가해 거기서 CCustomWizard 클래스의 인스턴스를 작성해 DoModal()에 의해 독자적인 위저드 모드 프롭퍼티 시트를 표시합니다. 덧붙여 표시한 위저드 모드의 프롭퍼티 시트를 [완료]버튼을 눌러 닫으면 DoModal()는 IDOK를 돌려줍니다.   　마지막으로 CommDlg.cpp 에서 maindlg.h 헤더의 앞에 CustomWizard.h 헤더를 인클루드 합니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | Wizard97스타일 |
| SetHeader()나 SetWatermark()를 호출하면 PSH\_WIZARD97플래그가 설정되어 위저드를 Wizard97 스타일로 할 수가 있습니다.   　다음에 나타내는 것은 방금 전 작성한 위저드 클래스를 Wizard97스타일에 고쳐 쓰는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl15-02/04.png  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl15-02/05.png   |  | | --- | | // CustomWizard.h  class CCustomPage : public CPropertyPageImpl<CCustomPage>  {  CButton m\_button\_topmost;  public:  enum { IDD = IDD\_PROPPAGE\_CUSTOM };  // constructor  CCustomPage(ATL::\_U\_STRINGorID title = (LPCTSTR)NULL)  : CPropertyPageImpl<CCustomPage>(title)  {  // 헤더 타이틀을 설정  SetHeaderTitle(\_T("최초의 페이지"));  SetHeaderSubTitle(\_T("부제"));  }  BEGIN\_MSG\_MAP\_EX(CCustomPage)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_CHECK\_NEXT, OnCheckNext)  CHAIN\_MSG\_MAP(CPropertyPageImpl<CCustomPage>)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  m\_button\_topmost = GetDlgItem(IDC\_CHECK\_NEXT);  return TRUE;  }  void OnCheckNext(UINT uNotifyCode, int nID, HWND hWndCtl){  OnSetActive();  }  BOOL OnSetActive(){  // 체크 박스가 체크되고 있는 경우는[다음에]버튼을 유효하게 해 ,  // 그렇지 않으면 무효로 한다. [돌아오는]버튼은 항상 무효.  SetWizardButtons(m\_button\_topmost.GetCheck() ? PSWIZB\_NEXT : 0);  return TRUE;  }  };  class CFinishPage : public CPropertyPageImpl<CFinishPage>  {  public:  enum { IDD = IDD\_PROPPAGE\_FINISH };  // constructor  CFinishPage(ATL::\_U\_STRINGorID title = (LPCTSTR)NULL)  : CPropertyPageImpl<CFinishPage>(title)  {  // 워터 마크를 표시하기 위해서 헤더를 비표시(?/font>  m\_psp.dwFlags |= PSP\_HIDEHEADER;  }  BEGIN\_MSG\_MAP\_EX(CFinishPage)  CHAIN\_MSG\_MAP(CPropertyPageImpl<CFinishPage>)  END\_MSG\_MAP()  BOOL OnSetActive(){  // [돌아오는]버튼과[완료]버튼을 유효하게 한다.  // 덧붙여[다음에]버튼은[완료]버튼으로 옮겨진다.  SetWizardButtons(PSWIZB\_BACK | PSWIZB\_FINISH);  return TRUE;  }  };  class CCustomWizard : public CPropertySheetImpl<CCustomWizard>  {  public:  CCustomPage m\_pageCustom;  CFinishPage m\_pageFinish;  CBitmap bmpHeader; // 헤더용 비트 맵  CBitmap bmpWatermark; // 워터 마크용 비트 맵  // constructor  CCustomWizard(ATL::\_U\_STRINGorID title = (LPCTSTR)NULL,  UINT uStartPage = 0, HWND hWndParent = NULL)  : CPropertySheetImpl<CCustomWizard>(title, uStartPage, hWndParent)  {  // 헤더용 비트 맵을 설정  bmpHeader.LoadBitmap(IDB\_BITMAP\_HEADER);  SetHeader(bmpHeader);  // 워터 마크용 비트 맵을 설정  bmpWatermark.LoadBitmap(IDB\_BITMAP\_WATERMARK);  SetWatermark(bmpWatermark);  AddPage(m\_pageCustom);  AddPage(m\_pageFinish);  }  BEGIN\_MSG\_MAP\_EX(CCustomWizard)  CHAIN\_MSG\_MAP(CPropertySheetImpl<CCustomWizard>)  END\_MSG\_MAP()  }; |   우선 헤더용으로 다음과 같은 비트맵 리소스를 프로젝트에 추가해 ID를 IDB\_BITMAP\_HEADER로 합니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl15-02/06.png  그리고 워터 마크(틈새를 만들어 )용으로 다음과 같은 비트맵 리소스를 프로젝트에 추가해 ID를 IDB\_BITMAP\_WATERMARK로 합니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl15-02/07.png  이러한 비트맵 리소스는 CCustomWizard 클래스의 constructor으로 위저드로 설정합니다.   위저드의 1페이지째에는 헤더 타이틀을 표시합니다. 그 때문에 CCustomPage 클래스의 constructor 　 으로 SetHeaderTitle()와 SetHeaderSubTitle()를 호출합니다.   위저드의 2페이지 째에는 워터 마크를 표시합니다. 워터 마크를 표시하기 위해서는 헤더를 비 표시로 할 필요가 있기때문에 CFinishPage 클래스의 constructor으로 PSP\_HIDEHEADER 플래그를 설정합니다. | |

▣ 인쇄 > 프린터 디바이스 컨텍스트

**< 프린트 디바이스 컨텍스트 >**  
  
WTL은 프린터 디바이스 컨텍스트를 CPrinterDC 클래스에서 캡슐화하고 있습니다. 이 클래스는 다른 디바이스 컨텍스트 클래스와 같게 CDC 클래스로부터 파생하고 있습니다.   
  
　이하에 나타내는 것은 CPrinterDC 클래스를 사용해 디폴트 프린터의 디바이스 컨텍스트를 취득하여 용지의 중앙에 「Hello, ATL/WTL」 이라고 하는 캐릭터 라인을 인쇄하는 예 입니다. 메뉴로부터 [인쇄]를 실행하면 인쇄를 개시합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h>  #include <atlprint.h> // CPrinterDC클래스를 사용하기 위해(때문에) |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("WTL:PrintSample"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT, OnFilePrint)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  void Draw(HDC hDC, CRect& rect){  CDCHandle dc(hDC);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  Draw(dc, rect);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnFilePrint(UINT uNotifyCode, int nID, HWND hWndCtl){  CPrinterDC dc;  DOCINFO di = {sizeof(di), \_T("WTL에 의한 인쇄 테스트")};  if(dc.StartDoc(&di) > 0){  if(dc.StartPage() > 0){  CRect rect(0, 0, dc.GetDeviceCaps(HORZRES), dc.GetDeviceCaps(VERTRES));  Draw(dc, rect);  if(dc.EndPage() > 0)  dc.EndDoc();  }  }  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 도구모음 자원의 [파일] 이하에 [인쇄] 메뉴 아이템을 추가하여 ID로서ID\_FILE\_PRINT를 설정합니다.   
  
　다음으로 stdafx.h 에서는 CPrinterDC 클래스를 사용하기 위해서 atlprint.h 헤더를 인클루드 합니다.   
  
　CMainFrame 클래스에서는 일반적인 프레임 윈도우를 작성하여 WM\_PAINT 메세지 핸들러를 추가해 클라이언트 영역의 중앙에 캐릭터 라인을 묘화 합니다. 이 때 묘화 코드를 Draw() 라고 하는 독자적인 함수로 정리합니다. Draw()는 인쇄 코드와 공용합니다.   
  
　메뉴 아이템 [인쇄](ID：ID\_FILE\_PRINT) 용의 커멘드 메세지 핸들러를 추가합니다. 이 핸들러에서는 CPrinterDC 클래스의 인스턴스를 작성해 디폴트 프린터의 디바이스 컨텍스트를 취득하여 Draw()를 호출해 용지의 중앙에 캐릭터 라인을 인쇄합니다.   
  
　이와 같이 CPrinterDC 클래스의 디폴트 constructor을 호출하면 디폴트 프린터의 디바이스 컨텍스트를 취득할 수 있습니다. 한편 CPrinterDC 클래스는 지정한 프린터의 디바이스 컨텍스트를 취득하기 위해서 인수 있어 constructor을 준비해 있습니다.

|  |
| --- |
| // atlprint.h  // CPrinterDC의 인수 있어 constructor  CPrinterDC(HANDLE hPrinter, const DEVMODE\* pdm = NULL)  {  CPrinterHandle p;  p.Attach(hPrinter);  Attach(p.CreatePrinterDC(pdm));  ATLASSERT(m\_hDC != NULL);  } |

이 constructor은 프린터의 핸들을 필요로 합니다. WTL은 프린터의 핸들을 CPrinterT 라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는 bool 값으로 true의 경우는 소멸자로 ClosePrinter()를 호출해 false의 경우는 호출하지 않습니다. atlprint.h헤더에서는typedef에 의해 다음과 같이 선언되고 있습니다.

|  |
| --- |
| // atlprint.h  typedef CPrinterT<false> CPrinterHandle;  typedef CPrinterT<true> CPrinter; |

　이하에 나타내는 것은 CPrinter 클래스를 사용해 프린터의 핸들을 취득하여 그 프린터의 핸들로부터 디바이스 컨텍스트를 취득해 인쇄하는 예입니다. 이번에 작성한 프로그램 OnFilePrint()를 다음과 같이 고쳐 씁니다.

|  |
| --- |
| // 지정한 이름의 프린터 핸들을 취득  CPrinter printer;  if(!printer.OpenPrinter(\_T("EPSON MJ-800C ESC/P R"))){  MessageBox(\_T("프린터 핸들의 취득에 실패했습니다. "));  return;  }  // 지정한 프린터의 프린터 디바이스 콘텍스트를 취득  CPrinterDC dc(printer);  DOCINFO di = {sizeof(di), \_T("WTL에 의한 인쇄 테스트")};  if(dc.StartDoc(&di) > 0){  if(dc.StartPage() > 0){  CRect rect(0, 0, dc.GetDeviceCaps(HORZRES), dc.GetDeviceCaps(VERTRES));  Draw(dc, rect);  if(dc.EndPage() > 0)  dc.EndDoc();  }  } |

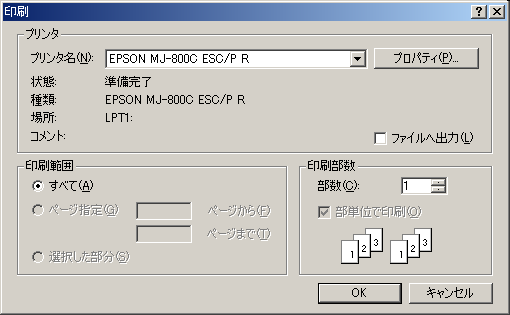
　이와 같이 CPrinterDC 클래스나 CPrinter 클래스를 사용하면 디바이스 컨텍스트나 프린터의 핸들을 명시적으로 삭제하거나 닫거나 할 필요가 없어지므로 편리합니다.

▣ 인쇄 > 인쇄 다이얼로그

**< 인쇄 다이얼로그 >**  
  
전회는 지정한 프린터의 디바이스 컨텍스트를 취득하기 위해서 직접 프린터명을 지정했습니다만 , 인쇄 다이얼로그를 사용하면 유저가 선택한 프린터의 디바이스 컨텍스트를 취득할 수가 있습니다. WTL은 common dialog의 인쇄 다이얼로그를 CPrintDialogImpl 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 이 클래스는 ATL의 CDialogImpl 클래스와 같이 base class로서 보고 사용합니다. atldlgs.h 헤더에서는 CPrintDialogImpl 클래스의 상속 클래스로서 CPrintDialog 클래스가 정의되고 있습니다.

|  |
| --- |
| // atldlgs.h  class CPrintDialog : public CPrintDialogImpl<CPrintDialog>  {  ...  ... |

　WTL의 CPrintDialog 클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 전회 작성한 프로그램을 고쳐 써 CPrintDialog 클래스로부터 프린터 디바이스 컨텍스트를 취득하는 예입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h>  #include <atldlgs.h> // CPrintDialog클래스를 사용하기 위해(때문에)  #include <atlprint.h> |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  CPrinter m\_printerCur; // 현재의 프린터 핸들  CDevMode m\_devmodeCur; // 현재의 프린터의 설정  public:  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("WTL:PrintSample"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  // constructor  CMainFrame(){  // 디폴트 프린터를 오픈  m\_printerCur.OpenDefaultPrinter();  m\_devmodeCur.CopyFromPrinter(m\_printerCur);  }  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT, OnFilePrint)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  void Draw(HDC hDC, CRect& rect){  CDCHandle dc(hDC);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  Draw(dc, rect);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnFilePrint(UINT uNotifyCode, int nID, HWND hWndCtl){  CPrintDialog dlg;  // 다이얼로그를 초기화  dlg.m\_pd.hDevMode = m\_devmodeCur.CopyToHDEVMODE();  dlg.m\_pd.hDevNames = m\_printerCur.CopyToHDEVNAMES();  if(dlg.DoModal() == IDOK){  // 프린터의 설정을 보존  m\_devmodeCur.CopyFromHDEVMODE(dlg.m\_pd.hDevMode);  // 현재의 프린터 핸들을 클로우즈 해 , 새로운 핸들을 오픈  m\_printerCur.ClosePrinter();  m\_printerCur.OpenPrinter(dlg.m\_pd.hDevNames, m\_devmodeCur);  CPrinterDC dc(m\_printerCur, m\_devmodeCur);  DOCINFO di = {sizeof(di), \_T("WTL에 의한 인쇄 테스트")};  if(dc.StartDoc(&di) > 0){  if(dc.StartPage() > 0){  CRect rect(0, 0,  dc.GetDeviceCaps(HORZRES), dc.GetDeviceCaps(VERTRES));  Draw(dc, rect);  if(dc.EndPage() > 0)  dc.EndDoc();  }  }  }  // 다이얼로그의 초기화로 할당한 메모리를 해방  ::GlobalFree(dlg.m\_pd.hDevMode);  ::GlobalFree(dlg.m\_pd.hDevNames);  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 stdafx.h 에서는 CPrintDialog 클래스를 사용하기 위해서 atldlgs.h 헤더를 인클루드 합니다.   
  
　다음에 CPrinter 클래스와 CDevMode 클래스의 인스턴스를 CMainFrame 클래스의 멤버 변수로서 선언합니다. 이러한 멤버 변수는 각각 현재의 프린터 핸들과 프린터의 설정을 보관 유지해 CMainFrame 클래스의 constructor으로 초기화 합니다.   
  
　다음에 메뉴 아이템 [인쇄](ID：ID\_FILE\_PRINT)용의 커멘드 메세지 핸들러를 변경합니다. 이 핸들러에서는 우선 CPrintDialog 클래스의 인스턴스를 작성합니다. CPrintDialog 클래스의 constructor인수는 모두 생략 할 수 있습니다만 , 제 1인수에 TRUE를 지정하면 [인쇄] 다이얼로그는 아니고 [프린터의 설정] 다이얼로그를 표시할 수 있습니다. 제 2인수는 플래그, 제 3인수는 부모 윈도우의 핸들을 지정할 수 있습니다.   
  
CPrintDialog 클래스의 인스턴스를 작성한 후 인쇄 다이얼로그를 초기화하기 위해서 다이얼로그의 m\_pd.hDevMode 과 m\_pd.hDevNames에 현재의 설정의 카피를 대입합니다. 이 때 CopyToHDEVMODE() 라고 CopyToHDEVNAMES()는 내부에서 GlobalAlloc()를 호출하고 있기 때문에 불필요하게 되었을 때에 GlobalFree를 호출할 필요가 있습니다.   
  
인쇄 다이얼로그를 표시하기 위해서는 DoModal()을 호출합니다. [OK] 버튼을 눌러 인쇄 다이얼로그를 닫았을 경우는 반환 값으로서 IDOK가 [캔슬]을 눌러 닫았을 경우는IDCANCEL 이 반환니다.   
  
IDOK 가 반환되어 왔을 경우는 우선 CopyFromHDEVMODE()을 호출해 m\_devmodeCur 현재의 프린터의 설정을 카피합니다. 다음에 현재의 프린터를 클로우즈 하여 인쇄 다이얼로그로 선택된 프린터를 오픈합니다. 프린터 핸들을 취득할 수 있으면 프린터 디바이스 콘텍스트를 작성하여 인쇄를 개시합니다.   
  
마지막으로 다이얼로그의 m\_pd.hDevMode 과 m\_pd.hDevNames에 대해서 GlobalFree를 호출합니다.   
  
　이와 같이 이 프로그램은 인쇄 다이얼로그를 닫을 때마다 프린터 핸들과 프린터의 설정을 CMainFrame 클래스의 멤버 변수에 보존하여 재차 인쇄 다이얼로그를 열 때에 전회 보존한 내용으로 인쇄 다이얼로그를 초기화하는 것으로 프린터 핸들과 프린터의 설정을 사용해 돌립니다.

▣ 인쇄 > 인쇄 프로퍼티 쉬트

**< 인쇄 프롭퍼티 시트 >**  
  
Windows 2000 이후에서는 인쇄 다이얼로그 대신에 인쇄 프롭퍼티 시트를 사용할 수 있습니다. WTL은 인쇄 프롭퍼티 시트를 CPrintDialogExImpl 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 이 클래스는 ATL의 CDialogImpl 클래스와 같이 base class로서 보고 사용합니다. atldlgs.h 헤더에서는 CPrintDialogExImpl 클래스의 상속 클래스로서 CPrintDialogEx 클래스가 정의 되고 있습니다.

|  |
| --- |
| // atldlgs.h  class CPrintDialogEx : public CPrintDialogExImpl<CPrintDialogEx>  {  ...  ... |

　WTL의 CPrintDialogEx 클래스의 사용법은 CPrintDialog 클래스와 거의 같습니다. 이하에 나타내는 것은 전회 작성한 프로그램을 고쳐 써 CPrintDialogEx 클래스로부터 프린터 디바이스 컨텍스트를 취득하는 예입니다.



|  |
| --- |
| // stdafx.h  #define WINVER 0x0500  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h>  #include <atldlgs.h> // CPrintDialogEx클래스를 사용하기 위해(때문에)  #include <atlprint.h> |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  CPrinter m\_printerCur; // 현재의 프린터 핸들  CDevMode m\_devmodeCur; // 현재의 프린터의 설정  public:  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("WTL:PrintSample"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  // constructor  CMainFrame(){  // 디폴트 프린터를 오픈  m\_printerCur.OpenDefaultPrinter();  m\_devmodeCur.CopyFromPrinter(m\_printerCur);  }  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT, OnFilePrint)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  void Draw(HDC hDC, CRect& rect){  CDCHandle dc(hDC);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  Draw(dc, rect);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnFilePrint(UINT uNotifyCode, int nID, HWND hWndCtl){  CPrintDialogEx dlg;  // 다이얼로그를 초기화  dlg.m\_pdex.hDevMode = m\_devmodeCur.CopyToHDEVMODE();  dlg.m\_pdex.hDevNames = m\_printerCur.CopyToHDEVNAMES();  if(SUCCEEDED(dlg.DoModal())){  // [인쇄]버튼 또는[적용]버튼을 눌렀을 경우  if(dlg.m\_pdex.dwResultAction == PD\_RESULT\_PRINT ||  dlg.m\_pdex.dwResultAction == PD\_RESULT\_APPLY)  {  // 프린터의 설정을 보존  m\_devmodeCur.CopyFromHDEVMODE(dlg.m\_pdex.hDevMode);  // 현재의 프린터 핸들을 클로우즈 해 , 새로운 핸들을 오픈  m\_printerCur.ClosePrinter();  m\_printerCur.OpenPrinter(dlg.m\_pdex.hDevNames, m\_devmodeCur);  }  // [인쇄]버튼을 눌렀을 경우  if(dlg.m\_pdex.dwResultAction == PD\_RESULT\_PRINT){  CPrinterDC dc(m\_printerCur, m\_devmodeCur);  DOCINFO di = {sizeof(di), \_T("WTL에 의한 인쇄 테스트")};  if(dc.StartDoc(&di) > 0){  if(dc.StartPage() > 0){  CRect rect(0, 0,  dc.GetDeviceCaps(HORZRES), dc.GetDeviceCaps(VERTRES));  Draw(dc, rect);  if(dc.EndPage() > 0)  dc.EndDoc();  }  }  }  }  // 다이얼로그의 초기화로 할당한 메모리를 해방  ::GlobalFree(dlg.m\_pdex.hDevMode);  ::GlobalFree(dlg.m\_pdex.hDevNames);  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 stdafx.h 에서는 CPrintDialogEx 클래스를 사용하기 위해서 WINVER를 0x0500에 정의하고 나서 atldlgs.h 헤더를 인클루드 합니다.   
  
　다음에 메뉴 아이템 [인쇄](ID：ID\_FILE\_PRINT) 용의 커멘드 메세지 핸들러를 변경합니다. 이 핸들러 에서는 우선 CPrintDialogEx 클래스의 인스턴스를 작성합니다. CPrintDialogEx 클래스의 constructor인수는 모두 생략 할 수 있습니다만 , 제1인수에는 플래그, 제2인수에는 부모 윈도우의 핸들을 지정할 수 있습니다. 인스턴스를 작성한 후 는 다이얼로그를 초기화 합니다.   
  
인쇄 프롭퍼티 시트를 표시하기 위해서는 DoModal()을 사용합니다. 이 함수가 성공했을 경우는 반환 값으로서 S\_OK가 돌아가 인쇄 프롭퍼티 시트로 누른 버튼에 의해 PRINTDLGEX 형태의 멤버 변수 m\_pdex의 dwResultAction 멤버에 다음의 값이 설정됩니다.

|  |  |
| --- | --- |
| 버튼명 | dlg.m\_pdex.dwResultAction의 값 |
| [인쇄] | PD\_RESULT\_PRINT |
| [캔슬] | PD\_RESULT\_CANCEL |
| [적용]을 누르고 나서[캔슬] | PD\_RESULT\_APPLY |

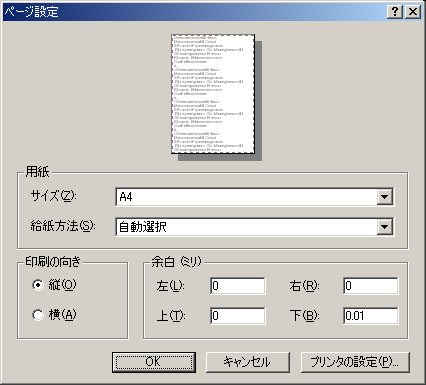
이번 예 에서는 [인쇄] 버튼을 눌러 인쇄 프롭퍼티 시트를 닫았을 경우는 프린터의 설정을 보존하고 나서 새로운 프린터 핸들을 오픈하여 한층 더 프린터 디바이스 컨텍스트를 취득해 인쇄를 개시합니다. [적용] 버튼을 누르고 나서 [캔슬] 버튼을 눌러 인쇄 프롭퍼티 시트를 닫았을 경우는 단지 프린터의 설정을 보존해 새로운 프린터 핸들을 오픈 하고 있습니다.

▣ 인쇄 > 페이지 설정 다이얼로그

**< 페이지 설정 다이얼로그 >**CPrintDialog 클래스의 constructor의 제 1인수에 TRUE를 지정하면 [프린터의 설정] 다이얼로그를 표시할 수 있습니다만 MSDN에 의하면 [페이지 설정] 다이얼로그의 사용을 추천할 수 있고 있습니다. WTL은 페이지 설정 다이얼로그를 CPageSetupDialogImpl 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 이 클래스는 ATL의CDialogImpl 클래스와 같이 base class로서 보고 사용합니다. atldlgs.h 헤더에서는 CPageSetupDialogImpl 클래스의 상속 클래스로서 CPageSetupDialog 클래스가 정의되고 있습니다.

|  |
| --- |
| // atldlgs.h  class CPageSetupDialog : public CPageSetupDialogImpl<CPageSetupDialog>  {  ...  ... |

　WTL의 CPageSetupDialog 클래스는 MFC의 동명의 클래스와 거의 같은 멤버 함수를 준비해 있습니다. 이하에 나타내는 것은 이전 작성한 인쇄 다이얼로그를 사용하는 프로그램에 페이지 설정 다이얼로그를 여는 기능을 추가하는 예입니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h>  #include <atldlgs.h> // CPrintDialog,CPageSetupDialog클래스를 사용하기 위해(때문에)  #include <atlprint.h> |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  CPrinter m\_printerCur; // 현재의 프린터 핸들  CDevMode m\_devmodeCur; // 현재의 프린터의 설정  public:  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("WTL:PrintSample"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  // constructor  CMainFrame(){  // 디폴트 프린터를 오픈  m\_printerCur.OpenDefaultPrinter();  m\_devmodeCur.CopyFromPrinter(m\_printerCur);  }  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT, OnFilePrint)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT\_SETUP, OnFilePrintSetup)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  void Draw(HDC hDC, CRect& rect){  CDCHandle dc(hDC);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  Draw(dc, rect);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnFilePrint(UINT uNotifyCode, int nID, HWND hWndCtl){  CPrintDialog dlg;  // 다이얼로그를 초기화  dlg.m\_pd.hDevMode = m\_devmodeCur.CopyToHDEVMODE();  dlg.m\_pd.hDevNames = m\_printerCur.CopyToHDEVNAMES();  if(dlg.DoModal() == IDOK){  // 프린터의 설정을 보존  m\_devmodeCur.CopyFromHDEVMODE(dlg.m\_pd.hDevMode);  // 현재의 프린터 핸들을 클로우즈 해 , 새로운 핸들을 오픈  m\_printerCur.ClosePrinter();  m\_printerCur.OpenPrinter(dlg.m\_pd.hDevNames, m\_devmodeCur);  CPrinterDC dc(m\_printerCur, m\_devmodeCur);  DOCINFO di = {sizeof(di), \_T("WTL에 의한 인쇄 테스트")};  if(dc.StartDoc(&di) > 0){  if(dc.StartPage() > 0){  CRect rect(0, 0,  dc.GetDeviceCaps(HORZRES), dc.GetDeviceCaps(VERTRES));  Draw(dc, rect);  if(dc.EndPage() > 0)  dc.EndDoc();  }  }  }  // 다이얼로그의 초기화로 할당한 메모리를 해방  ::GlobalFree(dlg.m\_pd.hDevMode);  ::GlobalFree(dlg.m\_pd.hDevNames);  }  void OnFilePrintSetup(UINT uNotifyCode, int nID, HWND hWndCtl){  CPageSetupDialog dlg;  // 다이얼로그를 초기화  dlg.m\_psd.hDevMode = m\_devmodeCur.CopyToHDEVMODE();  dlg.m\_psd.hDevNames = m\_printerCur.CopyToHDEVNAMES();  if(dlg.DoModal() == IDOK){  // 프린터의 설정을 보존  m\_devmodeCur.CopyFromHDEVMODE(dlg.m\_psd.hDevMode);  // 현재의 프린터 핸들을 클로우즈 해 , 새로운 핸들을 오픈  m\_printerCur.ClosePrinter();  m\_printerCur.OpenPrinter(dlg.m\_psd.hDevNames, m\_devmodeCur);  }  // 다이얼로그의 초기화로 할당한 메모리를 해방  ::GlobalFree(dlg.m\_psd.hDevMode);  ::GlobalFree(dlg.m\_psd.hDevNames);  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 도구모음 자원의 [파일] 이하에 [페이지 설정] 메뉴 아이템을 추가해 ID로 ID\_FILE\_PRINT\_SETUP를 설정합니다.   
  
　다음에 이 ID용의 커멘드 메세지 핸들러를 추가합니다. 이 핸들러 에서는 우선 CPageSetupDialog 클래스의 인스턴스를 작성합니다. CPageSetupDialog 클래스의 constructor의 인수는 모두 생략 할 수 있습니다만 제 1인수에는 플래그, 제 2인수에는 부모 윈도우의 핸들을 지정할 수 있습니다. 인스턴스를 작성한 후 는 다이얼로그를 초기화합니다.   
  
페이지 설정 다이얼로그를 표시하기 위해서는 DoModal()을 사용합니다. [OK] 버튼을 눌러 페이지 설정 다이얼로그를 닫았을 경우는 반환 값으로서 IDOK가 [캔슬]을 눌러 닫았을 경우는 IDCANCEL이 되돌아 옵니다.   
  
IDOK가 되돌아 왔을 경우는 GetPaperSize()을 호출해 용지 사이즈를 취득하거나 GetMargins()를 호출해 마진의 설정을 취득할 수가 있습니다. 이번 예 에서는 단지 프린터의 설정을 보존해 새로운 프린터 핸들을 오픈 하고 있습니다.

▣ 인쇄 > 인쇄 JOB

**< 인쇄 작업 >**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 지금까지의 인쇄 프로그램은 인쇄 코드를 다음과 같이 쓰고 있었습니다.   |  | | --- | | CPrinterDC dc(m\_printerCur, m\_devmodeCur);  DOCINFO di = {sizeof(di), \_T("WTL에 의한 인쇄 테스트")};  if(dc.StartDoc(&di) > 0){  if(dc.StartPage() > 0){  CRect rect(0, 0, dc.GetDeviceCaps(HORZRES), dc.GetDeviceCaps(VERTRES));  Draw(dc, rect);  if(dc.EndPage() > 0)  dc.EndDoc();  }  } |   WTL에서는 일반적인 인쇄 처리(인쇄 작업)를 CPrintJob 클래스와 CPrintJobInfo 클래스에서 캡슐화하고 있습니다. 이러한 클래스를 사용하면 StartDoc()나 StartPage() 를 스스로 쓸 필요는 없습니다. 그리고 인쇄 처리를 다른 thread로 실시하도록 지정할 수가 있습니다. 이하에 나타내는 것은 전회 작성한 프로그램의 인쇄 처리 코드를 CPrintJob클래스와 CPrintJobInfo 클래스를 사용해 고쳐 쓰는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl16-05/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h>  #include <atldlgs.h>  #include <atlprint.h> |  |  | | --- | | // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>, public CUpdateUI<CMainFrame>,  public CMessageFilter, public CIdleHandler, public CPrintJobInfo  {  CPrinter m\_printerCur; // 현재의 프린터 핸들  CDevMode m\_devmodeCur; // 현재의 프린터의 설정  CPrintJob m\_job; // 인쇄 작업  public:  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("WTL:PrintSample"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  // constructor  CMainFrame(){  // 디폴트 프린터를 오픈  m\_printerCur.OpenDefaultPrinter();  m\_devmodeCur.CopyFromPrinter(m\_printerCur);  }  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT, OnFilePrint)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT\_SETUP, OnFilePrintSetup)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  virtual bool IsValidPage(UINT nPage){  // 최초의 페이지만 인쇄  return (nPage == 0);  }  virtual bool PrintPage(UINT nPage, HDC hDC){  // 최초의 페이지만 인쇄  if(nPage >= 1)  return false;  CRect rect(0, 0,  ::GetDeviceCaps(hDC, HORZRES), ::GetDeviceCaps(hDC, VERTRES));  Draw(hDC, rect);  return true;  }  void Draw(HDC hDC, CRect& rect){  CDCHandle dc(hDC);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  Draw(dc, rect);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnFilePrint(UINT uNotifyCode, int nID, HWND hWndCtl){  CPrintDialog dlg;  // 다이얼로그를 초기화  dlg.m\_pd.hDevMode = m\_devmodeCur.CopyToHDEVMODE();  dlg.m\_pd.hDevNames = m\_printerCur.CopyToHDEVNAMES();  if(dlg.DoModal() == IDOK){  // 프린터의 설정을 보존  m\_devmodeCur.CopyFromHDEVMODE(dlg.m\_pd.hDevMode);  // 현재의 프린터 핸들을 클로우즈 해 , 새로운 핸들을 오픈  m\_printerCur.ClosePrinter();  m\_printerCur.OpenPrinter(dlg.m\_pd.hDevNames, m\_devmodeCur);  // 인쇄 작업을 개시  m\_job.StartPrintJob(false, m\_printerCur, m\_devmodeCur, this,  \_T("WTL에 의한 인쇄 테스트"), 0, 0, (dlg.PrintToFile() != FALSE));  }  // 다이얼로그의 초기화로 할당한 메모리를 해방  ::GlobalFree(dlg.m\_pd.hDevMode);  ::GlobalFree(dlg.m\_pd.hDevNames);  }  void OnFilePrintSetup(UINT uNotifyCode, int nID, HWND hWndCtl){  CPageSetupDialog dlg;  // 다이얼로그를 초기화  dlg.m\_psd.hDevMode = m\_devmodeCur.CopyToHDEVMODE();  dlg.m\_psd.hDevNames = m\_printerCur.CopyToHDEVNAMES();  if(dlg.DoModal() == IDOK){  // 프린터의 설정을 보존  m\_devmodeCur.CopyFromHDEVMODE(dlg.m\_psd.hDevMode);  // 현재의 프린터 핸들을 클로우즈 해 , 새로운 핸들을 오픈  m\_printerCur.ClosePrinter();  m\_printerCur.OpenPrinter(dlg.m\_psd.hDevNames, m\_devmodeCur);  }  // 다이얼로그의 초기화로 할당한 메모리를 해방  ::GlobalFree(dlg.m\_psd.hDevMode);  ::GlobalFree(dlg.m\_psd.hDevNames);  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 CPrintJobInfo 클래스를 CMainFrame 클래스의 base class에 추가합니다. 이 CPrintJobInfo 클래스에는 몇 개의 가상 함수가 준비되어 있어 CPrintJobInfo 클래스의 상속 클래스는 그것들을 오버라이드(override) 해 인쇄 처리를 기술합니다. CPrintJobInfo클래스가 준비해 있는 가상 함수를 이하에 나타냅니다.   |  |  | | --- | --- | | 함수명 | 해설 | | BeginPrintJob | 인쇄 처리를 개시하기 전(StartDoc()가 불려 간 직후) 의 처리 | | EndPrintJob | 인쇄 처리를 종료한 후 (EndDoc()가 불려 가기 직전) 의 처리 | | PrePrintPage | 페이지를 인쇄하기 전(StartPage()가 불려 간 직후) 의 처리 | | PostPrintPage | 페이지를 인쇄한 후 (EndPage()가 불려 가기 직전) 의 처리 | | PrintPage | 페이지의 인쇄 처리 | | GetNewDevModeForPage | 페이지 독자적인DEVMODE에의 포인터를 취득 | | IsValidPage | 페이지가 유효한지 어떤지를 지시 |   이 안에서 PrintPage()는 순수 가상 함수이기 때문에 반드시 오버라이드(override) 할 필요가 있습니다. 이번 예 에서는 CPrintJobInfo 클래스의 상속 클래스인 CMainFrame 클래스에서 IsValidPage()라고 PrintPage()를 오버라이드(override) 합니다.   IsValidPage()은 페이지 번호가 0의 경우만 true을 돌려주어 인쇄 작업을 계속하는 것을 지시합니다. 이것은 이번 프로그램은 최초의 페이지 밖에 인쇄하지 않기 때문입니다.   PrintPage()은 현재의 페이지 번호를 확인해 Draw()를 호출해 인쇄를 실시하고 있습니다.   　다음에 CPrintJob 클래스의 인스턴스를 CMainFrame 클래스의 멤버 변수로서 선언합니다.   　다음에 메뉴 아이템 [인쇄](ID：ID\_FILE\_PRINT) 용의 커멘드 메세지 핸들러를 변경합니다. 이 핸들러 에서는 CPrintJob 클래스의 멤버 함수인 StartPrintJob()을 호출해 인쇄 작업을 개시합니다. StartPrintJob()의 인수는 순서에 인쇄 작업을 위해서 thread를 기동하는지 어떤지를 나타내는 bool값, 프린터 핸들 DEVMODE에의 포인터, IPrintJobInfo 클래스에의 포인터, 문서명, 개시 페이지 번호, 종료 페이지 번호, 파일에 출력하는지 어떤지를 나타내는 bool값, 출력 파일명을 지정할 수 있습니다. 덧붙여 제 4인수의 IPrintJobInfo 클래스란 CPrintJobInfo 클래스의 base class 입니다. 이번 예에서는 CMainFrame 클래스는 CPrintJobInfo 클래스의 상속 클래스이기 때문에 CMainFrame 클래스는 간접적으로 IPrintJobInfo클래스의 상속 클래스라는 것이 됩니다. |

▣ 인쇄 > 인쇄 미리보기

**< 인쇄 미리 보기 >**　WTL은 인쇄 프리뷰용 윈도우를 CPrintPreviewWindowImpl 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 이 클래스는 ATL의CDialogImpl 클래스와 같이 base class 로서 사용합니다. atlprint.h 헤더에서는 CPrintPreviewWindowImpl 클래스의 상속 클래스로서 CPrintPreviewWindow 클래스가 정의되고 있습니다.

|  |
| --- |
| // atlprint.h  class CPrintPreviewWindow : public CPrintPreviewWindowImpl<CPrintPreviewWindow>  {  ...  ... |

　이하에 나타내는 것은 전회 작성한 프로그램을 고쳐 써, 인쇄 프리뷰 기능을 추가하는 예입니다. 메뉴의 [인쇄 프리뷰]를 실행할 때마다 인쇄 프리뷰 윈도우의 표시/비표시를 새로 바꿉니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h>  #include <atldlgs.h>  #include <atlprint.h> |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>, public CUpdateUI<CMainFrame>,  public CMessageFilter, public CIdleHandler, public CPrintJobInfo  {  CPrinter m\_printerCur; // 현재의 프린터 핸들  CDevMode m\_devmodeCur; // 현재의 프린터의 설정  CPrintJob m\_job; // 인쇄 작업  CPrintPreviewWindow m\_wndPreview; // 인쇄 프리뷰 윈도우  public:  DECLARE\_FRAME\_WND\_CLASS\_EX(\_T("WTL:PrintSample"), IDR\_MAINFRAME,  CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS, COLOR\_WINDOW)  // constructor  CMainFrame(){  // 디폴트 프린터를 오픈  m\_printerCur.OpenDefaultPrinter();  m\_devmodeCur.CopyFromPrinter(m\_printerCur);  }  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  UPDATE\_ELEMENT(ID\_FILE\_PRINT\_PREVIEW, UPDUI\_MENUPOPUP)  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT, OnFilePrint)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT\_PREVIEW, OnFilePrintPreview)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_PRINT\_SETUP, OnFilePrintSetup)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  virtual bool IsValidPage(UINT nPage){  // 최초의 페이지만 인쇄  return (nPage == 0);  }  virtual bool PrintPage(UINT nPage, HDC hDC){  // 최초의 페이지만 인쇄  if(nPage >= 1)  return false;  CRect rect(0, 0,  ::GetDeviceCaps(hDC, HORZRES), ::GetDeviceCaps(hDC, VERTRES));  Draw(hDC, rect);  return true;  }  void Draw(HDC hDC, CRect& rect){  CDCHandle dc(hDC);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  Draw(dc, rect);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  UISetCheck(ID\_FILE\_PRINT\_PREVIEW, false);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnFilePrint(UINT uNotifyCode, int nID, HWND hWndCtl){  CPrintDialog dlg;  // 다이얼로그를 초기화  dlg.m\_pd.hDevMode = m\_devmodeCur.CopyToHDEVMODE();  dlg.m\_pd.hDevNames = m\_printerCur.CopyToHDEVNAMES();  if(dlg.DoModal() == IDOK){  // 프린터의 설정을 보존  m\_devmodeCur.CopyFromHDEVMODE(dlg.m\_pd.hDevMode);  // 현재의 프린터 핸들을 클로우즈 해 , 새로운 핸들을 오픈  m\_printerCur.ClosePrinter();  m\_printerCur.OpenPrinter(dlg.m\_pd.hDevNames, m\_devmodeCur);  // 인쇄 작업을 개시  m\_job.StartPrintJob(false, m\_printerCur, m\_devmodeCur, this,  \_T("WTL에 의한 인쇄 테스트"), 0, 0, (dlg.PrintToFile() != FALSE));  }  // 다이얼로그의 초기화로 할당한 메모리를 해방  ::GlobalFree(dlg.m\_pd.hDevMode);  ::GlobalFree(dlg.m\_pd.hDevNames);  }  void OnFilePrintPreview(UINT uNotifyCode, int nID, HWND hWndCtl){  static bool bPreview = false;  if(bPreview){  // 인쇄 프리뷰 윈도우를 파기  m\_hWndClient = NULL;  m\_wndPreview.DestroyWindow();  }else{  // 인쇄 프리뷰 윈도우를 작성  m\_wndPreview.SetPrintPreviewInfo(m\_printerCur, m\_devmodeCur, this, 0, 0);  m\_wndPreview.SetPage(0);  m\_wndPreview.Create(m\_hWnd, rcDefault, NULL, 0, WS\_EX\_CLIENTEDGE);    m\_hWndClient = m\_wndPreview;  }  bPreview = !bPreview;  UISetCheck(ID\_FILE\_PRINT\_PREVIEW, bPreview);  UpdateLayout();  }  void OnFilePrintSetup(UINT uNotifyCode, int nID, HWND hWndCtl){  CPageSetupDialog dlg;  // 다이얼로그를 초기화  dlg.m\_psd.hDevMode = m\_devmodeCur.CopyToHDEVMODE();  dlg.m\_psd.hDevNames = m\_printerCur.CopyToHDEVNAMES();  if(dlg.DoModal() == IDOK){  // 프린터의 설정을 보존  m\_devmodeCur.CopyFromHDEVMODE(dlg.m\_psd.hDevMode);  // 현재의 프린터 핸들을 클로우즈 해 , 새로운 핸들을 오픈  m\_printerCur.ClosePrinter();  m\_printerCur.OpenPrinter(dlg.m\_psd.hDevNames, m\_devmodeCur);  }  // 다이얼로그의 초기화로 할당한 메모리를 해방  ::GlobalFree(dlg.m\_psd.hDevMode);  ::GlobalFree(dlg.m\_psd.hDevNames);  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 도구모음 자원의 [파일] 아래에 [인쇄 프리뷰] 메뉴 아이템을 추가하여 ID 로서 ID\_FILE\_PRINT\_PREVIEW를 설정합니다.   
  
　다음으로 CPrintPreviewWindow 클래스의 인스턴스를 CMainFrame 클래스의 멤버 변수로서 선언합니다.   
  
　다음으로 메뉴 아이템 [인쇄 프리뷰](ID：ID\_FILE\_PRINT\_PREVIEW) 용의 커멘드 메세지 핸들러를 추가합니다. 이 핸들러 에서는 우선 bPreview이라고 하는 정적 변수를 선언합니다. 이 변수는 인쇄 프리뷰 윈도우 상태를 보관 유지합니다.   
  
인쇄 프리뷰 윈도우를 작성하기 위해서는 우선 SetPrintPreviewInfo()을 호출합니다. 이 함수의 인수는 순서에 프린터 핸들 DEVMODE 구조체에의 포인터 IPrintJobInfo 클래스에의 포인터, 최소 페이지, 최대 페이지를 지정할 수 있습니다. 다음에 SetPage()를 호출해 최초로 표시하는 페이지를 지정하여 Create()를 호출해 인쇄 프리뷰 윈도우를 작성합니다.

▣ 뷰 > 스크롤

**< 스크롤 >**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 프레임 윈도우의 뷰로써 스크롤 윈도우를 작성합니다. WTL은 스크롤 윈도우를 작성하기 위해서 CScrollWindowImpl라고 하는 템플릿 클래스를 준비해 있습니다. 이 클래스는ATL의 CWindowImpl 클래스와 같이 base class로서의 보고 사용하여 템플릿 인수의 종류도 CWindowImpl클래스와 같습니다.   　이하에 나타내는 것은 CScrollWindowImpl 클래스를 프레임 윈도우의 뷰로써 사용하는 예입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl17-01/01.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h>  #include <atlscrl.h> // 스크롤 윈도우 클래스를 사용하기 위해(때문에) |  |  | | --- | | // view.h  class CScrollView : public CScrollWindowImpl<CScrollView>  {  public:  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CScrollView)  MSG\_WM\_CREATE(OnCreate)  CHAIN\_MSG\_MAP(CScrollWindowImpl<CScrollView>)  CHAIN\_MSG\_MAP\_ALT(CScrollWindowImpl<CScrollView>, 1)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetScrollSize(640, 480);  SetScrollLine(10, 10);  SetScrollPage(100, 100);  SetMsgHandled(false);  return lRet;  }  void DoPaint(CDCHandle dc){  // 스크롤 윈도우의 중앙에 캐릭터 라인을 묘화  CSize size;  GetScrollSize(size);  CRect rect(0, 0, size.cx, size.cy);  dc.DrawText(\_T("Hello, ATL/WTL"),  -1, rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  }; |  |  | | --- | | // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CScrollView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN,  WS\_EX\_CLIENTEDGE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "view.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 stdafx.h 에서는 CScrollWindowImpl 클래스를 사용하기 위해서 atlscrl.h 헤더를 인클루드 합니다.   　다음에 뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 view.h라고 하는 헤더 파일을 추가하여 거기에 CScrollWindowImpl클래스로부터 파생했다. CScrollView 라고 하는 클래스를 정의하고 있습니다.   　CScrollView 클래스 내에서는 메세지 맵을 준비하여 WM\_CREATE 메세지 엔트리와 base class에의 체인을 추가합니다. WM\_CREATE 메세지 핸들러에서는 스크롤 윈도우의 사이즈나 스크롤 바의 이동량을 설정하여 SetMsgHandled(false)를 호출합니다. 이것은 base class에서도 WM\_CREATE 메세지 핸들러가 준비되어 있기 때문입니다.   　다음으로 base class의 멤버 함수 DoPaint()를 오버라이드(override) 합니다. 이 함수는 CScrollWindowImpl 클래스에서 준비되어 있는 WM\_PAINT메세지 핸들러로 부터 불려져 여기에 뷰의 묘화 코드를 기술하는 것으로 스크롤 바에 맞추어 묘화 내용도 자동적으로 스크롤 되게 됩니다. 위의 예에서는 스크롤 윈도우의 중앙에 캐릭터 라인을 묘화 하고 있습니다.   　이와 같이 CScrollWindowImpl 클래스에는 메세지 맵이 준비되어 있어 WM\_CREATE 메시지나 WM\_PAINT메세지 그 외 WM\_VSCROLL, WM\_HSCROLL, WM\_MOUSEWHEEL, WM\_SIZE, WM\_SETTINGCHANGE, WM\_PRINTCLIENT 메세지를 매핑 하고 있기 때문에 적은 작업으로 기본적인 기능을 가진 스크롤 윈도우를 작성할 수가 있습니다.   　그런데 CScrollWindowImpl클래스는 대체 메세지 맵도 준비해 있습니다. 이것은 뷰를 상하 좌우에 스크롤하기 위한 커멘드 메세지를 매핑 하고 있습니다. WTL은 스크롤 윈도우를 조작하기 위해서 디폴트로 다음과 같은 ID를 준비해 있습니다.   |  |  | | --- | --- | | ID | 조작 | | ID\_SCROLL\_UP | 상에 스크롤 | | ID\_SCROLL\_DOWN | 하에 스크롤 | | ID\_SCROLL\_PAGE\_UP | 전의 페이지 | | ID\_SCROLL\_PAGE\_DOWN | 다음의 페이지 | | ID\_SCROLL\_TOP | 최상부 | | ID\_SCROLL\_BOTTOM | 최하부 | | ID\_SCROLL\_LEFT | 왼쪽으로 스크롤 | | ID\_SCROLL\_RIGHT | 오른쪽으로 스크롤 | | ID\_SCROLL\_PAGE\_LEFT | 왼쪽의 페이지 | | ID\_SCROLL\_PAGE\_RIGHT | 오른쪽의 페이지 | | ID\_SCROLL\_ALL\_LEFT | 왼쪽의 단 | | ID\_SCROLL\_ALL\_RIGHT | 우의 구석 |   이번 예 에서는 메뉴에ID\_SCROLL\_UP, ID\_SCROLL\_DOWN,, D\_SCROLL\_LEFT, ID\_SCROLL\_RIGHT의 4개의 아이템을 추가했습니다.   http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl17-01/02.png  이것에 의해 이러한 메뉴 아이템을 선택하면 커멘드 체인에 의해 커멘드 메세지가 프레임 윈도우로부터 뷰 윈도우에 전송 되어 한층 더 대체 메세지 맵에의 체인에 의해 CScrollWindowImpl클래스의 대체 메세지 맵에 전송 됩니다. CScrollWindowImpl클래스에서는 간접적으로DoScroll()라고 하는 멤버 함수가 불려져 그 함수 내에서 ID마다 스크롤 처리를 합니다. |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 매핑 방식 스크롤 윈도우 |
| WTL은 매핑 방식에 대응한 스크롤 윈도우용으로 CMapScrollWindowImpl클래스를 준비해 있습니다. 이 클래스는 CScrollWindowImpl클래스에 매핑방식용의 기능이 추가된 것으로 사용법도 CScrollWindowImpl 클래스와 거의 같습니다.   　이하에 나타내는 것은 CMapScrollWindowImpl 클래스로부터 파생한 CMapScrollView 클래스를 작성하는 예입니다.   |  | | --- | | // view.h  class CMapScrollView : public CMapScrollWindowImpl<CMapScrollView>  {  public:  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMapScrollView)  MSG\_WM\_CREATE(OnCreate)  CHAIN\_MSG\_MAP(CMapScrollWindowImpl<CMapScrollView>)  CHAIN\_MSG\_MAP\_ALT(CMapScrollWindowImpl<CMapScrollView>, 1)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetScrollMapMode(MM\_LOMETRIC); // 매핑방식을 설정  SetScrollSize(640, 480);  SetScrollLine(10, 10);  SetScrollPage(100, 100);  SetMsgHandled(false);  return lRet;  }  void DoPaint(CDCHandle dc){  // 스크롤 윈도우의 중앙에 캐릭터 라인을 묘화  CRect rect;  GetScrollSize(rect);  dc.DrawText(\_T("Hello, ATL/WTL"),  -1, rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  }; |   CMapScrollWindowImpl 클래스의 디폴트의 매핑방식은 MM\_TEXT입니다. CMapScrollView 클래스에서는 매핑방식을 MM\_LOMETRIC로 설정해 있기 때문에 스크롤 윈도우의 논리 단위는 0.1mm에 맵 됩니다 | |

|  |  |
| --- | --- |
| http://home.att.ne.jp/banana/akatsuki/img/square.png | 플랫 스크롤 윈도우 |
| WTL은 공용 컨트롤의 플랫 스크롤 바에 대응한 스크롤 윈도우용으로 CFSBWindowT라고 하는 템플릿 클래스를 준비해 있습니다. 템플릿 인수에는 클래스를 지정합니다만 그 클래스는 CFSBWindowT 클래스의 base class인 CFlatScrollBarImpl 클래스의 템플릿 인수로서 사용됩니다. atlscrl.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlscrl.h  template <class TBase = ATL::CWindow>  class CFSBWindowT : public TBase, public CFlatScrollBarImpl<CFSBWindowT< TBase > >  {  ...  };  typedef CFSBWindowT<ATL::CWindow> CFSBWindow; |   　이하에 나타내는 것은 CScrollWindowImpl클래스로부터 파생한 CFlatScrollView 클래스를 작성하는 예입니다. CFSBWindow 클래스는 CScrollWindowImpl 클래스의 제2템플릿 인수로 지정합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl17-01/03.png   |  | | --- | | // view.h  class CFlatScrollView : public CScrollWindowImpl<CFlatScrollView, CFSBWindow>  {  public:  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 스크롤 사이즈  enum {SCROLL\_CX = 640, SCROLL\_CY = 480};  // 메세지 맵  typedef CScrollWindowImpl<CFlatScrollView, CFSBWindow> baseClass;  BEGIN\_MSG\_MAP\_EX(CFlatScrollView)  MSG\_WM\_SIZE(OnSize)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_DESTROY(OnDestroy)  CHAIN\_MSG\_MAP(baseClass)  CHAIN\_MSG\_MAP\_ALT(baseClass, 1)  END\_MSG\_MAP()  void OnSize(UINT uType, CSize size){  if(size.cx < SCROLL\_CX)  FlatSB\_ShowScrollBar(SB\_HORZ);  if(size.cy < SCROLL\_CY)  FlatSB\_ShowScrollBar(SB\_VERT);  SetMsgHandled(false);  }  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  FlatSB\_Initialize();  SetScrollSize(SCROLL\_CX, SCROLL\_CY);  SetScrollLine(10, 10);  SetScrollPage(100, 100);  SetMsgHandled(false);  return lRet;  }  void OnDestroy(){  FlatSB\_Uninitialize();  }  void DoPaint(CDCHandle dc){  // 스크롤 윈도우의 중앙에 캐릭터 라인을 묘화  CSize size;  GetScrollSize(size);  CRect rect(0, 0, size.cx, size.cy);  dc.DrawText(\_T("Hello, ATL/WTL"),  -1, rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  }; |   CFSBWindow 클래스를 사용하기 위해서는 atlscrl.h 헤더의 앞에 atlctrls.h 헤더를 인클루드 할 필요가 있습니다. CFlatScrollView 클래스내에서는 WM\_CREATE 메세지 핸들러와 WM\_DESTROY 메세지 핸들러로 각각 플랫 툴바의 초기화와 종료 처리를 실시하고 있습니다.   덧붙여 이대로는 윈도우를 스크롤 사이즈보다 크게 하고 나서 다시 되돌렸을 경우 스크롤 바가 안보이게 되어 버립니다. 그런데 이번 예에서는 WM\_SIZE 메세지 핸들러를 추가해 클라이언트 영역이 스크롤 사이즈보다 작은 경우에 스크롤 바를 가시화하고 있습니다. WTL7.5.4291그리고 추가된 SCRL\_DISABLENOSCROLL확장 스타일을 SetScrollExtendedStyle()로 설정하는 것으로 WM\_SIZE 메세지 핸들러를 준비할 필요가 없어졌습니다. | |

▣ 뷰 > 리스트 박스

**< 리스트 박스 >**  
  
이하에 나타내는 것은 리스트 박스 컨트롤을 베이스로 한 뷰 윈도우를 작성하는 예입니다. 뷰 윈도우에는 사용 가능한 TrueType폰트를 열거합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // view.h  class CListboxView : public CWindowImpl<CListboxView, CListBox>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CListBox::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CListboxView)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_CODE\_HANDLER\_EX(LBN\_DBLCLK, OnListDblClick)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetFont(AtlGetDefaultGuiFont());  CClientDC dc(m\_hWnd);  EnumFontFamilies(dc, NULL,  (FONTENUMPROC)FontProc, (LPARAM)this);  return lRet;  }  static int CALLBACK FontProc(ENUMLOGFONT \*lpelf,  NEWTEXTMETRIC \*lpntm, int nFontType, LPARAM lParam)  {  CListboxView\* pList = (CListboxView\*)lParam;  // TrueType폰트명만 리스트에 추가  if(nFontType & TRUETYPE\_FONTTYPE)  pList->AddString(lpelf->elfLogFont.lfFaceName);  return 1;  }  void OnListDblClick(UINT uNotifyCode, int nID, HWND hWndCtl){  // 현재 선택되고 있는 아이템을 취득해 표시  int nIndex = GetCurSel();  if(nIndex != LB\_ERR){  CString strText;  GetText(nIndex, strText);  MessageBox(strText);  }  }  }; |

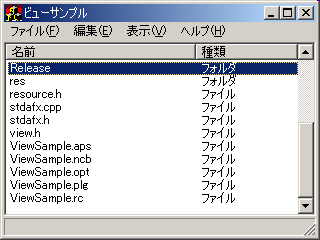
|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CListboxView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN | WS\_VSCROLL |  LBS\_NOINTEGRALHEIGHT | LBS\_NOTIFY | LBS\_WANTKEYBOARDINPUT | LBS\_SORT,  WS\_EX\_CLIENTEDGE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "view.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 stdafx.h 에서는 컨트롤 용 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   
  
　다음으로 뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 view.h 라고 하는 헤더 파일을 추가하여 거기에 CWindowImpl 클래스로부터 파생했다. CListboxView라고 하는 클래스를 정의하고 있습니다. CWindowImpl 클래스의 제2인수에는 베이스가 되는 CListBox 클래스를 지정합니다.   
  
　CListboxView 클래스 내에서는 우선 WM\_CREATE 메세지 핸들러를 추가하여 초에DefWindowProc()를 호출하고 나서 디폴트의 GUI 폰트를 설정해 뷰에 폰트명을 열거합니다.   
  
　다음에 통지 코드가 LBN\_DBLCLK의 커멘드 메세지 핸들러를 추가합니다. 여기에서는 선택되고 있는 폰트명을 취득하여 메시지 박스로 표시하고 있을 뿐입니다.

▣ 뷰 > 리스트 뷰

**< 리스트 뷰 >**  
  
이하에 나타내는 것은 리스트 뷰 컨트롤을 베이스로 한 뷰 윈도우를 작성하는 예입니다. 뷰 윈도우에는 현재 디렉토리의 폴더명과 파일명을 열거합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // view.h  class CListView : public CWindowImpl<CListView, CListViewCtrl>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CListViewCtrl::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CListView)  MSG\_WM\_CREATE(OnCreate)  REFLECTED\_NOTIFY\_CODE\_HANDLER\_EX(NM\_DBLCLK, OnListDblClick)  DEFAULT\_REFLECTION\_HANDLER()  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetFont(AtlGetDefaultGuiFont());  SetExtendedListViewStyle(LVS\_EX\_INFOTIP | LVS\_EX\_FULLROWSELECT);  // 리스트뷰에 컬럼 삽입  CRect rcList;  ::GetClientRect(GetParent(), rcList);  int nScrollWidth = GetSystemMetrics(SM\_CXVSCROLL);  int n3DEdge = GetSystemMetrics(SM\_CXEDGE);  InsertColumn(0, \_T("이름"), LVCFMT\_LEFT, 190, -1);  InsertColumn(1, \_T("종류"), LVCFMT\_LEFT,  rcList.Width() - 190 - nScrollWidth - n3DEdge \* 2, -1);  // 리스트뷰에 아이템 추가  CFindFile find;  if(find.FindFile()){  do{  if(!find.IsDots()){  int nIndex = GetItemCount();  AddItem(nIndex, 0, find.GetFileName());  AddItem(nIndex, 1,  find.IsDirectory() ? \_T("폴더") : \_T("파일"));  }  }while(find.FindNextFile());  }  return lRet;  }  LRESULT OnListDblClick(LPNMHDR pnmh){  LPNMITEMACTIVATE pnmia = (LPNMITEMACTIVATE)pnmh;  CString strItem;  GetItemText(pnmia->iItem, 0, strItem);  CString strSubItem;  GetItemText(pnmia->iItem, 1, strSubItem);  MessageBox(strItem + \_T("은") + strSubItem + \_T("입니다. "), \_T("아이템"));  return 0;  }  }; |

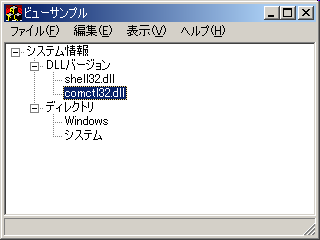
|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CListView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  REFLECT\_NOTIFICATIONS() // 메세지 리플렉션  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN |  LVS\_REPORT | LVS\_SHOWSELALWAYS,  WS\_EX\_CLIENTEDGE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "view.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 stdafx.h 에서는 컨트롤용 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   
  
　다음에 뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 view.h 라고 하는 헤더 파일을 추가하여 거기에 CWindowImpl 클래스로부터 파생한 CListView 라고 하는 클래스를 정의하고 있습니다. CWindowImpl 클래스의 제2인수에는 베이스가 되는CListViewCtrl 클래스를 지정합니다.   
  
　CListView 클래스 내에서는 우선 WM\_CREATE 메세지 핸들러를 추가하여 최초에DefWindowProc()를 호출하고 나서 디폴트의 GUI폰트를 설정해 뷰에 현재 디렉토리의 폴더명과 파일명을 열거합니다.   
  
　다음에 통지 코드가 NM\_DBLCLK의 통지 메세지 핸들러를 추가합니다. 여기에서는 선택되고 있는 아이템 캐릭터 라인을 취득하여 메시지 박스로 표시하고 있을 뿐입니다. 덧붙여 이 함수는 메세지 리플렉션에 의해 불려 집니다.

▣ 뷰 > 트리 뷰

**< 트리 뷰 >**  
  
이하에 나타내는 것은 트리뷰 컨트롤을 베이스로 한 뷰 윈도우를 작성하는 예입니다. 뷰 윈도우에는 아이템을 추가하여 아이템을 더블 클릭 하면 그 아이템에 관한 정보를 메시지 박스에 표시합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // view.h  class CTreeView : public CWindowImpl<CTreeView, CTreeViewCtrl>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CTreeViewCtrl::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  enum {SHELL = 1, COMCTL, WINDIR, SYSDIR}; // 트리 아이템 식별자  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CTreeView)  MSG\_WM\_CREATE(OnCreate)  REFLECTED\_NOTIFY\_CODE\_HANDLER\_EX(NM\_DBLCLK, OnTreeDblClick)  DEFAULT\_REFLECTION\_HANDLER()  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetFont(AtlGetDefaultGuiFont());  // 트리뷰에 아이템 삽입  HTREEITEM hSysInfo =  InsertItem(\_T("시스템 정보"), TVI\_ROOT, TVI\_LAST);  HTREEITEM hDllVer =  InsertItem(\_T("DLL버젼"), hSysInfo, TVI\_LAST);  HTREEITEM hItem;  hItem = InsertItem(\_T("shell32.dll"), hDllVer, TVI\_LAST);  SetItemData(hItem, SHELL);  hItem = InsertItem(\_T("comctl32.dll"), hDllVer, TVI\_LAST);  SetItemData(hItem, COMCTL);  HTREEITEM hSysDir =  InsertItem(\_T("디렉토리"), hSysInfo, TVI\_LAST);  hItem = InsertItem(\_T("Windows"), hSysDir, TVI\_LAST);  SetItemData(hItem, WINDIR);  hItem = InsertItem(\_T("시스템"), hSysDir, TVI\_LAST);  SetItemData(hItem, SYSDIR);  return lRet;  }  LRESULT OnTreeDblClick(LPNMHDR pnmh){  // 클릭된 아이템 핸들 취득  UINT uFlag;  CPoint pt = ::GetMessagePos();  ScreenToClient(&pt);  HTREEITEM hItem = HitTest(pt, &uFlag);  if(hItem == NULL || !(uFlag & TVHT\_ONITEM))  return 0;  // 아이템 데이터 취득  CString strInfo;  DWORD dwData = GetItemData(hItem);  switch(dwData){  case SHELL:  {  // shell32.dll의 버젼 취득  DWORD dwMajor, dwMinor;  AtlGetShellVersion(&dwMajor, &dwMinor);  strInfo.Format(\_T("Version %d.%02d"), dwMajor, dwMinor);  break;  }  case COMCTL:  {  // comctl32.dll의 버젼 취득  DWORD dwMajor, dwMinor;  AtlGetCommCtrlVersion(&dwMajor, &dwMinor);  strInfo.Format(\_T("Version %d.%02d"), dwMajor, dwMinor);  break;  }  case WINDIR:  {  // Windows폴더 패스 취득  TCHAR szPath[MAX\_PATH];  GetWindowsDirectory(szPath, MAX\_PATH);  strInfo = szPath;  break;  }  case SYSDIR:  {  // 시스템 폴더 패스 취득  TCHAR szPath[MAX\_PATH];  GetSystemDirectory(szPath, MAX\_PATH);  strInfo = szPath;  break;  }  default:  break;  }  if(!strInfo.IsEmpty())  MessageBox(strInfo, \_T("아이템"));  return 0;  }  }; |

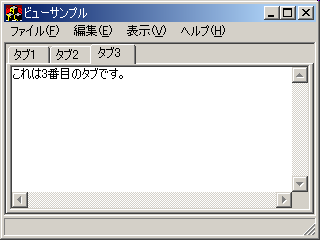
|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CTreeView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  REFLECT\_NOTIFICATIONS() // 메세지 리플렉션  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN |  TVS\_HASBUTTONS | TVS\_HASLINES | TVS\_LINESATROOT | TVS\_SHOWSELALWAYS,  WS\_EX\_CLIENTEDGE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "view.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 stdafx.h 에서는 컨트롤용 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   
  
　다음에 뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 view.h 라고 하는 헤더 파일을 추가하여 거기에 CWindowImpl 클래스로부터 파생한 CTreeView 라고 하는 클래스를 정의하고 있습니다. CWindowImpl 클래스의 제2인수에는 베이스가 되는CTreeViewCtrl 클래스를 지정합니다.   
  
　CTreeView 클래스 내에서는 우선 WM\_CREATE 메세지 핸들러를 추가하여 처음에DefWindowProc()를 호출하고 나서 디폴트의 GUI 폰트를 설정해 뷰에 트리 아이템을 삽입합니다.   
  
　다음에 통지 코드가 NM\_DBLCLK의 통지 메세지 핸들러를 추가합니다. 여기에서는 선택되고 있는 아이템에 관련 지을 수 있었던 수치에 의해 적절한 시스템 정보를 취득하여 메시지 박스로 표시하고 있습니다. 덧붙여 이 함수는 메세지 리플렉션에 의해 불려 갑니다.   
  
　이번 예에서는 뷰 윈도우의 베이스로서 CTreeViewCtrl 클래스를 사용했습니다만 WTL이 트리뷰 컨트롤을 위해서 준비해 있는 CTreeViewCtrlEx 클래스를 사용할 수도 있습니다.

▣ 뷰 > 탭

**< 탭 >**  
  
이하에 나타내는 것은 탭 컨트롤을 베이스로 한 뷰 윈도우를 작성하는 예입니다. 도구모음의 [파일] 로부터 [신규 작성]을 실행하면 탭을 추가하여 [닫는]을 실행하면 경향 탭(선택되고 있는 탭) 을 삭제합니다. 각 탭에는 에디트 컨트롤을 표시합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // view.h  class CTabView : public CWindowImpl<CTabView, CTabCtrl>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CTabCtrl::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CTabView)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_SIZE(OnSize)  MSG\_WM\_DESTROY(OnDestroy)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_NEW, OnFileNew)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_CLOSE, OnFileClose)  REFLECTED\_NOTIFY\_CODE\_HANDLER\_EX(TCN\_SELCHANGING, OnTabSelChanging)  REFLECTED\_NOTIFY\_CODE\_HANDLER\_EX(TCN\_SELCHANGE, OnTabSelChange)  DEFAULT\_REFLECTION\_HANDLER()  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetFont(AtlGetDefaultGuiFont());  return lRet;  }  void OnSize(UINT uType, CSize size){  // 경향 탭내의 에디트 컨트롤을 리사이즈  int nIndex = GetCurSel();  if(nIndex != -1)  ResizeEdit(nIndex);  SetMsgHandled(false);  }  void OnDestroy(){  // 모든 탭을 삭제  int nCount = GetItemCount();  for(int i=nCount-1; i>=0; i--)  RemoveTab(i);  SetMsgHandled(false);  }  void OnFileNew(UINT uNotifyCode, int nID, HWND hWndCtl){  // 탭을 추가  AddTab();  }  void OnFileClose(UINT uNotifyCode, int nID, HWND hWndCtl){  // 경향 탭을 삭제  int nIndex = GetCurSel();  if(nIndex != -1)  RemoveTab(nIndex);  }  LRESULT OnTabSelChanging(LPNMHDR pnmh){  // 경향 탭내의 에디트 컨트롤을 비표시로 하는  ShowCurEdit(false);  return 0;  }  LRESULT OnTabSelChange(LPNMHDR pnmh){  // 경향 탭내의 에디트 컨트롤을 표시하는  ShowCurEdit(true);  return 0;  }  void AddTab(){  // 경향 탭내의 에디트 컨트롤을 비표시로 하는  ShowCurEdit(false);  // 탭명을 작성  static int nTitle = 1;  CString strTitle;  strTitle.Format(\_T("탭%d"), nTitle++);  // 탭에 표시하는 에디트 컨트롤을 작성  CEdit\* pEdit = new CEdit;  pEdit->Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_HSCROLL | WS\_VSCROLL |  ES\_AUTOHSCROLL | ES\_AUTOVSCROLL | ES\_MULTILINE | ES\_NOHIDESEL | ES\_SAVESEL,  WS\_EX\_CLIENTEDGE);  pEdit->SetFont(AtlGetDefaultGuiFont());  // 탭을 삽입  TCITEM ti;  ti.mask = TCIF\_TEXT | TCIF\_PARAM;  ti.pszText = (LPTSTR)(LPCTSTR)strTitle;  ti.lParam = (LPARAM)pEdit;  int nIndex = InsertItem(GetItemCount(), &ti);  // 삽입한 탭을 경향으로 해 에디트 컨트롤에 포커스를 주는  SetCurSel(nIndex);  pEdit->SetFocus();  // 에디트 컨트롤을 리사이즈  ResizeEdit(nIndex);  }  void RemoveTab(int nIndex){  CEdit\* pEdit = GetEditCtrl(nIndex);  pEdit->DestroyWindow();  delete pEdit;  DeleteItem(nIndex);  // 경향 탭을 선두의 탭으로 변경  SetCurSel(0);  ShowCurEdit(true);  }  CEdit\* GetEditCtrl(int nIndex){  // 탭에 관련지을 수 있었던 에디트 컨트롤에의 포인터를 취득  TCITEM ti;  ti.mask = TCIF\_PARAM;  GetItem(nIndex, &ti);  return (CEdit\*)ti.lParam;  }  void ResizeEdit(int nIndex){  CRect rcTabClient;  GetClientRect(rcTabClient);  AdjustRect(FALSE, rcTabClient);  CEdit\* pEdit = GetEditCtrl(nIndex);  pEdit->MoveWindow(rcTabClient);  }  void ShowCurEdit(bool bShow){  int nIndex = GetCurSel();  if(nIndex != -1){  CEdit\* pEdit = GetEditCtrl(nIndex);  if(bShow){  ResizeEdit(nIndex);  pEdit->ShowWindow(SW\_SHOW);  pEdit->SetFocus();  }else{  pEdit->ShowWindow(SW\_HIDE);  }  }  }  }; |

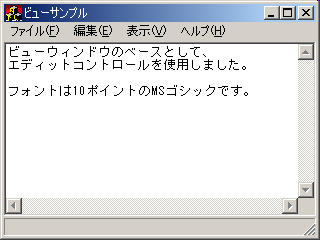
|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CTabView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  REFLECT\_NOTIFICATIONS() // 메세지 리플렉션  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN,  WS\_EX\_CLIENTEDGE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "view.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 도구 모음 자원의 [파일] 이하에 [신규 작성] 이라고 [닫는] 메뉴 아이템을 추가하여 ID 로서 각각 ID\_FILE\_NEWID\_FILE\_CLOSE를 설정합니다.   
  
　다음에 stdafx.h 에서는 컨트롤용 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   
  
　다음으로 뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 view.h 라고 하는 헤더 파일을 추가하여 거기에 CWindowImpl 클래스로부터 파생한 CTabView 라고 하는 클래스를 정의하고 있습니다. CWindowImpl 클래스의 제2인수에는 베이스가 되는CTabCtrl 클래스를 지정합니다.   
  
　CTabView 클래스 내에서는 탭과 탭 내에 표시하는 에디트 컨트롤을 조작하기 위해서 독자적인 멤버 함수를 몇 개인가 준비합니다. AddTab()는 탭을 추가합니다. 각 탭에는 에디트 컨트롤에의 포인터를 관련 짓습니다. RemoveTab()은 지정한 탭을 삭제합니다. 탭에 관련 지을 수 있었던 에디트 컨트롤에의 포인터도 삭제합니다. GetEditCtrl()은 지정한 탭에 관련 지을 수 있었던 에디트 컨트롤에의 포인터를 취득합니다. ResizeEdit()는 지정한 에디트 컨트롤의 사이즈를 탭 사이즈에 맞도록 변경합니다. ShowCurEdit()는 경향 탭내의 에디트 컨트롤의 표시/비표시를 새로 바꿉니다. 인수에 true를 지정했을 경우는 표시, false를 지정했을 경우는 비 표시로 합니다.   
  
　다음으로 WM\_CREATE 메세지 핸들러를 추가하여 처음에 DefWindowProc()를 호출하고 나서 디폴트의 GUI 폰트를 설정합니다.   
  
　다음에 WM\_SIZE 메세지 핸들러를 추가하여 ResizeEdit()를 호출해 윈도우 사이즈가 변경될 때마다 탭 내의 에디트 컨트롤의 사이즈를 변경합니다.   
  
　다음에 WM\_DESTROY 메세지 핸들러를 추가해 모든 탭에 대해서 RemoveTab()를 호출합니다.   
  
　다음에 ID가 ID\_FILE\_NEW의 커멘드 메세지 핸들러를 추가해 AddTab()를 호출해 탭을 추가합니다.   
  
　 ID가 ID\_FILE\_CLOSE의 커멘드 메세지 핸들러를 추가해 RemoveTab()를 호출해 현재의 탭을 삭제합니다.   
  
　 통지 코드가 TCN\_SELCHANGING의 탭 콘트롤용 통지 메세지 핸들러를 추가합니다. 이 메세지 핸들러는 탭이 바뀌기 직전에 불려 집니다만 여기에서는 ShowCurEdit()에 false를 지정해 호출하는 것으로 바뀌기 직전의 탭의 에디트 컨트롤을 비 표시로 합니다.   
  
통지 코드가 TCN\_SELCHANGE의 탭 콘트롤용 통지 메세지 핸들러를 추가합니다. 이 메세지 핸들러는 탭이 바뀐 직후에 불려 집니다만 여기에서는 ShowCurEdit()에 true를 지정해 호출하는 것으로 바뀐 직후의 탭의 에디트 컨트롤을 표시합니다.

▣ 뷰 > 에디터

**< 에디트 >**  
  
이하에 나타내는 것은 에디트 컨트롤을 베이스로 한 뷰 윈도우를 작성하는 예입니다. 뷰 윈도우의 폰트는 10포인트 사이즈의 MS고딕으로 설정합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // view.h  class CEditView : public CWindowImpl<CEditView, CEdit>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CEdit::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  CFont m\_font;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CEditView)  MSG\_WM\_CREATE(OnCreate)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  // 폰트 설정  m\_font.CreatePointFont(100, \_T("MS 고딕"));  SetFont(m\_font);  return lRet;  }  }; |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CEditView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN |  WS\_HSCROLL | WS\_VSCROLL | ES\_AUTOHSCROLL | ES\_AUTOVSCROLL |  ES\_MULTILINE | ES\_NOHIDESEL,  WS\_EX\_CLIENTEDGE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "view.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

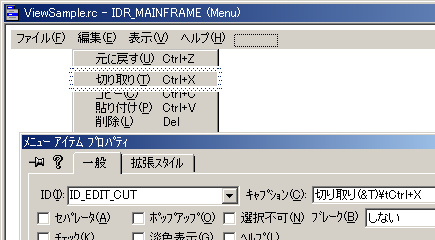
　우선 stdafx.h 에서는 컨트롤용 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   
  
　 뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 view.h라고 하는 헤더 파일을 추가하여 거기에 CWindowImpl클래스로부터 파생한 CEditView라고 하는 클래스를 정의하고 있습니다. CWindowImpl 클래스의 제2인수에는 베이스가 되는 CEdit 클래스를 지정합니다.   
  
　CEditView 클래스 내에서는 WM\_CREATE 메세지 핸들러를 추가하여 DefWindowProc()를 호출하고 나서 폰트를 설정합니다. 이번 예에서는 10포인트 사이즈의 MS고딕으로 설정해 있습니다.   
  
　그런데 WTL는 CEditCommands이라고 하는 클래스를 준비해 있습니다. 이 클래스는 에디트 컨트롤을 조작하기 위한 커멘드 메세지 핸들러와 UI 갱신 핸들러용의 멤버 함수를 준비해 있습니다. 다음에 나타내는 것은 앞의 프로그램에 CEditCommands클래스를 추가해 메뉴로부터 에디트 컨트롤을 조작하는 기능을 추가하는 예입니다. 덧붙여 변경하는 것은 CEditView 클래스와 CMainFrame 클래스입니다.

|  |
| --- |
| // view.h  class CEditView : public CWindowImpl<CEditView, CEdit>,  public CEditCommands<CEditView>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CEdit::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  CFont m\_font;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CEditView)  MSG\_WM\_CREATE(OnCreate)  CHAIN\_MSG\_MAP\_ALT(CEditCommands<CEditView>, 1)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  // 폰트 설정  m\_font.CreatePointFont(100, \_T("MS 고딕"));  SetFont(m\_font);  return lRet;  }  }; |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CEditView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIEnable(ID\_EDIT\_UNDO, m\_view.CanUndo());  UIEnable(ID\_EDIT\_CUT, m\_view.CanCut());  UIEnable(ID\_EDIT\_COPY, m\_view.CanCopy());  UIEnable(ID\_EDIT\_CLEAR, m\_view.CanClear());  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  UPDATE\_ELEMENT(ID\_EDIT\_UNDO, UPDUI\_MENUPOPUP)  UPDATE\_ELEMENT(ID\_EDIT\_CUT, UPDUI\_MENUPOPUP)  UPDATE\_ELEMENT(ID\_EDIT\_COPY, UPDUI\_MENUPOPUP)  UPDATE\_ELEMENT(ID\_EDIT\_CLEAR, UPDUI\_MENUPOPUP)  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN |  WS\_HSCROLL | WS\_VSCROLL | ES\_AUTOHSCROLL | ES\_AUTOVSCROLL |  ES\_MULTILINE | ES\_NOHIDESEL,  WS\_EX\_CLIENTEDGE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

　우선 CEditView 클래스의 base class에 CEditCommands 클래스를 추가해 메세지 맵에 CEditCommands 클래스의 대체 메세지 맵에의 체인을 추가합니다.   
  
WTL은 에디트 컨트롤을 조작하기 위해서 디폴트로 다음과 같은ID을 준비해 있습니다.

|  |  |
| --- | --- |
| ID | 조작 |
| ID\_EDIT\_CLEAR | 삭제 |
| ID\_EDIT\_CLEAR\_ALL | 모두 삭제 |
| ID\_EDIT\_COPY | 카피 |
| ID\_EDIT\_CUT | 절취 |
| ID\_EDIT\_PASTE | 붙이기 |
| ID\_EDIT\_SELECT\_ALL | 모두 선택 |
| ID\_EDIT\_UNDO | 바탕으로 되돌리는 |

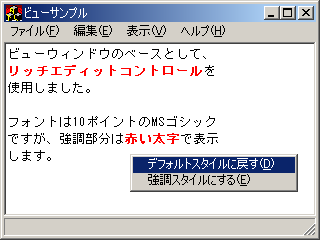
이 예에서는 메뉴에 ID\_EDIT\_UNDO,ID\_EDIT\_CUT, ID\_EDIT\_COPY, ID\_EDIT\_PASTE, ID\_EDIT\_CLEAR 의 5개의 아이템을 추가했습니다.   
  
  
  
이것에 의해 이러한 메뉴 아이템을 선택하면 커멘드 체인에 의해 커멘드 메세지가 프레임 윈도우로부터 뷰 윈도우에 전송 되어 한층 더 대체 메세지 맵에의 체인에 의해 CEditCommands 클래스의 대체 메세지 맵에 전송 됩니다. CEditCommands 클래스에서는 에디트 컨트롤을 조작하기 위한 함수가 불려 집니다.   
  
　다음에 CMainFrame 클래스의UI 갱신 핸들러 맵에 ID\_EDIT\_UNDO, ID\_EDIT\_CUT, ID\_EDIT\_COPY, ID\_EDIT\_CLEAR 용무의 엔트리를 추가해 OnIdle()안에서 그것들에 대한 UIEnable()을 호출합니다. 덧붙여 CEditCommands클래스는 에디트 컨트롤의 UI갱신 핸들러 용으로 다음과 같은 함수를 준비해 있습니다.

|  |  |
| --- | --- |
| 함수명 | 기능 |
| CanCut | 에디트 컨트롤에 선택된 캐릭터 라인이 있는지 어떤지 |
| CanCopy | 에디트 컨트롤에 선택된 캐릭터 라인이 있는지 어떤지 |
| CanClear | 에디트 컨트롤에 선택된 캐릭터 라인이 있는지 어떤지 |
| CanSelectAll | 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |
| CanFind | 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |
| CanRepeat | 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |
| CanReplace | 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |
| CanClearAll | 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |

CMainFrame 클래스에서는 아이들 시에 이러한 함수로 에디트 컨트롤의 내용을 조사해 메뉴 아이템 상태를 갱신하고 있습니다.

▣ 뷰 > 리치 에디터

**< 리치 에디트 >**  
  
이하에 나타내는 것은 리치 에디트 컨트롤을 베이스로 한 뷰 윈도우를 작성하는 예입니다. 뷰 윈도우의 폰트는 10포인트 사이즈의 MS고딕으로 설정합니다. 그리고 오른쪽 클릭한 후 표시되는 pop-up menu로부터 [강조 스타일로 하는]을 선택하면 선택 부분의 캐릭터 라인이 붉은 굵은 글씨가 됩니다.

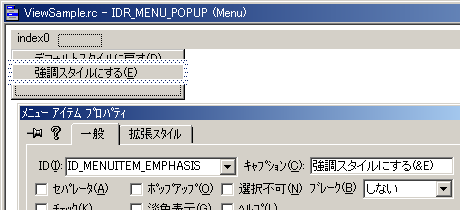


|  |
| --- |
| // stdafx.h  // 리치 에디트 컨트롤의 버젼1.0을 사용하는 것을 정의  #define \_RICHEDIT\_VER 0x0100  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // view.h  class CRichEditView : public CWindowImpl<CRichEditView, CRichEditCtrl>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CRichEditCtrl::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  CFont m\_font;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CRichEditView)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_DEFAULT, OnMenuDefault)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EMPHASIS, OnMenuEmphasis)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  // 폰트 설정  m\_font.CreatePointFont(100, \_T("MS 고딕"));  SetFont(m\_font);  return lRet;  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 선택 캐릭터 라인의 마지막에 설정  if(pt.x == -1 && pt.y == -1){  LONG start, end;  GetSel(start, end);  pt = PosFromChar(end);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  void OnMenuDefault(UINT uNotifyCode, int nID, HWND hWndCtl){  CString strText;  if(GetSelText(strText) != 0){  CHARFORMAT cf = {sizeof(CHARFORMAT)};  GetDefaultCharFormat(cf);  SetSelectionCharFormat(cf);  }  }  void OnMenuEmphasis(UINT uNotifyCode, int nID, HWND hWndCtl){  CString strText;  if(GetSelText(strText) != 0){  CHARFORMAT cf = {sizeof(CHARFORMAT)};  cf.dwMask = CFM\_COLOR | CFM\_BOLD;  cf.dwEffects = CFE\_BOLD;  cf.crTextColor = RGB(255, 0, 0);  SetSelectionCharFormat(cf);  }  }  }; |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CRichEditView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN |  WS\_HSCROLL | WS\_VSCROLL | ES\_AUTOHSCROLL | ES\_AUTOVSCROLL |  ES\_MULTILINE | ES\_NOHIDESEL | ES\_SAVESEL,  WS\_EX\_CLIENTEDGE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "view.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  // 리치 에디트 컨트롤 초기화  HINSTANCE hRich = LoadLibrary(CRichEditCtrl::GetLibraryName());  if(hRich == NULL){  AtlMessageBox(NULL, \_T("리치 에디트 컨트롤 초기화 실패"),  \_T("에러"), MB\_OK | MB\_ICONERROR);  return 0;  }  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  FreeLibrary(hRich);  return nRet;  } |

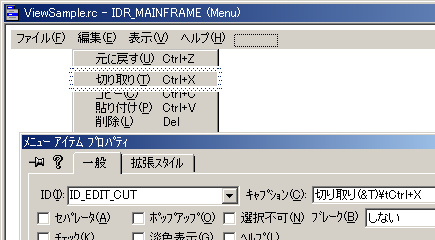
　우선 stdafx.h 에서는 컨트롤용 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다. 한층 더 버전 1.0의 리치 에디트 컨트롤을 사용하므로 \_RICHEDIT\_VER 를 0x0100 에 정의합니다.   
  
　뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 view.h라고 하는 헤더 파일을 추가해 거기에 CWindowImpl클래스로부터 파생한 CRichEditView라고 하는 클래스를 정의하고 있습니다. CWindowImpl클래스의 제2인수에는 베이스가 되는 CRichEditCtrl 클래스를 지정합니다.   
  
　CRichEditView 클래스 내에서는 WM\_CREATE 메세지 핸들러를 추가해 DefWindowProc()를 호출하고 나서 폰트를 설정합니다. 이번 예에서는 10 포인트 사이즈의 MS고딕으로 설정해 있습니다.   
  
　다음에 WM\_CONTEXTMENU 메세지 핸들러를 추가해 pop-up menu를 표시하기 위한 코드를 추가합니다. 덧붙여 메뉴 자원의 ID는 IDR\_MENU\_POPUP 으로 해 톱 레벨에 [index0]를 그 아래에 [디폴트 스타일에 되돌리는] 메뉴 아이템과 [강조 스타일로 하는] 메뉴 아이템을 추가합니다. 아이템에는 각각 ID\_MENUITEM\_DEFAULTID\_MENUITEM\_EMPHASIS라고 하는 ID을 설정합니다.   
  
  
  
　다음에 pop-up menu 아이템의 커멘드 메세지 핸들러를 추가합니다. [디폴트 스타일에 되돌리는]에서는 디폴트의 스타일로 설정해 [강조 스타일로 하는]에서는 붉은 굵은 글씨를 설정해 있습니다.   
  
　그런데 WTL은 CRichEditCommands이라고 하는 클래스를 준비해 있습니다. 이 클래스는 CEditCommands 클래스로부터 파생하고 있어 버전 2.0의 리치 에디트 컨트롤 용으로 ID\_EDIT\_REDO 라고 하는 커멘드 ID가 추가되고 있는 것을 제외해 사용법은 CEditCommands 클래스와 다르지 않습니다.   
  
　다음에 나타내는 것은 앞의 프로그램에 CRichEditCommands클래스를 추가해 메뉴로부터 리치 에디트 컨트롤을 조작하는 기능을 추가하는 예입니다. 덧붙여 변경하는 것은 CRichEditView 클래스와 CMainFrame클래스입니다.

|  |
| --- |
| // view.h  class CRichEditView : public CWindowImpl<CRichEditView, CRichEditCtrl>,  public CRichEditCommands<CRichEditView>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CRichEditCtrl::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  CFont m\_font;  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CRichEditView)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_CONTEXTMENU(OnContextMenu)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_DEFAULT, OnMenuDefault)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_EMPHASIS, OnMenuEmphasis)  CHAIN\_MSG\_MAP\_ALT(CRichEditCommands<CRichEditView>, 1)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  // 폰트 설정  m\_font.CreatePointFont(100, \_T("MS 고딕"));  SetFont(m\_font);  return lRet;  }  void OnContextMenu(HWND hWnd, CPoint pt){  // [Shift]+[F10]키가 밀렸을 경우는 좌표를 선택 캐릭터 라인의 마지막에 설정  if(pt.x == -1 && pt.y == -1){  LONG start, end;  GetSel(start, end);  pt = PosFromChar(end);  ClientToScreen(&pt);  }  // 좌표가 클라이언트 영역내의 경우만 pop-up menu를 표시  CRect rc;  GetClientRect(&rc);  ClientToScreen(&rc);  if(rc.PtInRect(pt)){  CMenu menuPopup;  menuPopup.LoadMenu(IDR\_MENU\_POPUP);  menuPopup.GetSubMenu(0).TrackPopupMenu(  TPM\_LEFTALIGN | TPM\_TOPALIGN | TPM\_LEFTBUTTON, pt.x, pt.y, m\_hWnd);  }else{  SetMsgHandled(false);  }  }  void OnMenuDefault(UINT uNotifyCode, int nID, HWND hWndCtl){  CString strText;  if(GetSelText(strText) != 0){  CHARFORMAT cf = {sizeof(CHARFORMAT)};  GetDefaultCharFormat(cf);  SetSelectionCharFormat(cf);  }  }  void OnMenuEmphasis(UINT uNotifyCode, int nID, HWND hWndCtl){  CString strText;  if(GetSelText(strText) != 0){  CHARFORMAT cf = {sizeof(CHARFORMAT)};  cf.dwMask = CFM\_COLOR | CFM\_BOLD;  cf.dwEffects = CFE\_BOLD;  cf.crTextColor = RGB(255, 0, 0);  SetSelectionCharFormat(cf);  }  }  }; |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CRichEditView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIEnable(ID\_EDIT\_UNDO, m\_view.CanUndo());  UIEnable(ID\_EDIT\_CUT, m\_view.CanCut());  UIEnable(ID\_EDIT\_COPY, m\_view.CanCopy());  UIEnable(ID\_EDIT\_CLEAR, m\_view.CanClear());  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  UPDATE\_ELEMENT(ID\_EDIT\_UNDO, UPDUI\_MENUPOPUP)  UPDATE\_ELEMENT(ID\_EDIT\_CUT, UPDUI\_MENUPOPUP)  UPDATE\_ELEMENT(ID\_EDIT\_COPY, UPDUI\_MENUPOPUP)  UPDATE\_ELEMENT(ID\_EDIT\_CLEAR, UPDUI\_MENUPOPUP)  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN |  WS\_HSCROLL | WS\_VSCROLL | ES\_AUTOHSCROLL | ES\_AUTOVSCROLL |  ES\_MULTILINE | ES\_NOHIDESEL | ES\_SAVESEL,  WS\_EX\_CLIENTEDGE);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

　우선 CRichEditView 클래스의 base class에 CRichEditCommands 클래스를 추가해 메세지 맵에 CRichEditCommands 클래스의 대체 메세지 맵에의 체인을 추가합니다.   
  
WTL은 리치 에디트 컨트롤을 조작하기 위해서 디폴트로 다음과 같은 ID를 준비해 있습니다.

|  |  |
| --- | --- |
| ID | 조작 |
| ID\_EDIT\_CLEAR | 삭제 |
| ID\_EDIT\_CLEAR\_ALL | 모두 삭제 |
| ID\_EDIT\_COPY | 카피 |
| ID\_EDIT\_CUT | 절취 |
| ID\_EDIT\_PASTE | 붙이기 |
| ID\_EDIT\_SELECT\_ALL | 모두 선택 |
| ID\_EDIT\_UNDO | 바탕으로 되돌리는 |
| ID\_EDIT\_REDO | 재시도 |

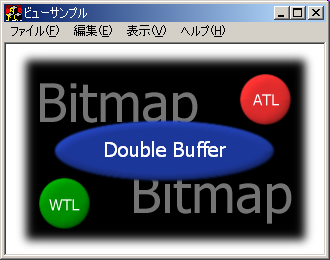
이 예에서는 메뉴에 ID\_EDIT\_UNDO, ID\_EDIT\_CUT, ID\_EDIT\_COPY, ID\_EDIT\_PASTE, ID\_EDIT\_CLEAR의 5개의 아이템을 추가했습니다.   
  
  
  
이것에 의해 이러한 메뉴 아이템을 선택하면 커멘드 체인에 의해 커멘드 메세지가 프레임 윈도우로부터 뷰 윈도우에 전송 되어 한층 더 대체 메세지 맵에의 체인에 의해 CRichEditCommands 클래스의 대체 메세지 맵에 전송 됩니다. CRichEditCommands 클래스에서는 리치 에디트 컨트롤을 조작하기 위한 함수가 불려 집니다.   
  
CMainFrame 클래스의 UI갱신 핸들러 맵에 ID\_EDIT\_UNDO, ID\_EDIT\_CUT, ID\_EDIT\_COPY, ID\_EDIT\_CLEAR 용무의 엔트리를 추가해 OnIdle()안에서 그것들에 대한 UIEnable()을 호출합니다. 덧붙여 CRichEditCommands 클래스는 리치 에디트 컨트롤의 UI갱신 핸들러 용으로 다음과 같은 함수를 준비해 있습니다.

|  |  |
| --- | --- |
| 함수명 | 기능 |
| CanCut | 리치 에디트 컨트롤에 선택된 캐릭터 라인이 있는지 어떤지 |
| CanCopy | 리치 에디트 컨트롤에 선택된 캐릭터 라인이 있는지 어떤지 |
| CanClear | 리치 에디트 컨트롤에 선택된 캐릭터 라인이 있는지 어떤지 |
| CanSelectAll | 리치 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |
| CanFind | 리치 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |
| CanRepeat | 리치 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |
| CanReplace | 리치 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |
| CanClearAll | 리치 에디트 컨트롤에 캐릭터 라인이 있는지 어떤지 |

CMainFrame 클래스에서는 아이들 시에 이러한 함수로 리치 에디트 컨트롤의 내용을 조사해 메뉴 아이템 상태를 갱신하고 있습니다.

▣ 뷰 > 폼

**< 폼 >**  
  
이하에 나타내는 것은 다이얼로그 윈도우(폼)를 베이스로 한 뷰 윈도우를 작성하는 예입니다. 뷰 윈도우의 상부에는 사용 가능한 TrueType 폰트를 열거한 리스트 박스 컨트롤을 배치해 폰트를 선택하면 뷰 윈도우의 하부(뷰 윈도우아래로부터 높이50의 구형) 에 그 폰트로 폰트 명을 묘화 합니다.

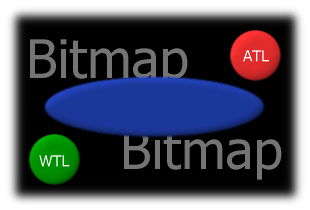


|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // view.h  class CFormView : public CDialogImpl<CFormView>  {  public:  enum { IDD = IDD\_FORMVIEW\_FONT };  BOOL PreTranslateMessage(MSG\* pMsg){  return CWindow::IsDialogMessage(pMsg);  }  private:  CListBox m\_list\_font;  CString m\_strFont; // 묘화 하는 폰트명  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CFormView)  MSG\_WM\_INITDIALOG(OnInitDialog)  MSG\_WM\_PAINT(OnPaint)  MSG\_WM\_SIZE(OnSize)  COMMAND\_CODE\_HANDLER\_EX(LBN\_SELCHANGE, OnListSelChange)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  m\_list\_font = GetDlgItem(IDC\_LIST\_FONT);  // 사용 가능한TrueType폰트명을 열거  CClientDC dc(m\_hWnd);  EnumFontFamilies(dc, NULL,  (FONTENUMPROC)FontProc, (LPARAM)m\_list\_font.m\_hWnd);  return TRUE;  }  static int CALLBACK FontProc(ENUMLOGFONT \*lpelf,  NEWTEXTMETRIC \*lpntm, int nFontType, LPARAM lParam)  {  CListBox list = (HWND)lParam;  // TrueType폰트명만 리스트에 추가  if(nFontType & TRUETYPE\_FONTTYPE)  list.AddString(lpelf->elfLogFont.lfFaceName);  return 1;  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  if(!m\_strFont.IsEmpty()){  // 사이즈20포인트의 폰트를 작성해 , 선택  CFont font;  font.CreatePointFont(200, m\_strFont);  HFONT hOldFont = dc.SelectFont(font);  // 뷰윈드우 하부에 폰트명을 묘화  CRect rcFontView = GetFontViewRect();  dc.SetBkMode(TRANSPARENT);  dc.DrawText(m\_strFont,  m\_strFont.GetLength(), rcFontView, DT\_VCENTER | DT\_SINGLELINE);  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  }  }  void OnSize(UINT uType, CSize size){  if(m\_list\_font.IsWindow()){  // 뷰윈드우 상부에 리스트 박스 컨트롤을 배치  CRect rcForm;  GetClientRect(rcForm);  rcForm.bottom -= 50;  m\_list\_font.MoveWindow(rcForm);  // 뷰윈드우 하부를 갱신  InvalidateRect(GetFontViewRect());  }  }  void OnListSelChange(UINT uNotifyCode, int nID, HWND hWndCtl){  // 현재 선택되고 있는 아이템을 취득해 표시  int nIndex = m\_list\_font.GetCurSel();  if(nIndex != LB\_ERR){  m\_list\_font.GetText(nIndex, m\_strFont);  // 뷰윈드우 하부를 갱신  InvalidateRect(GetFontViewRect());  }  }  CRect GetFontViewRect(){  // 폰트명을 묘화 하는 구형(뷰윈드우아래로부터 높이50의 구형) 을 취득  CRect rcForm;  GetClientRect(rcForm);  rcForm.top = rcForm.bottom - 50;  return rcForm;  }  }; |

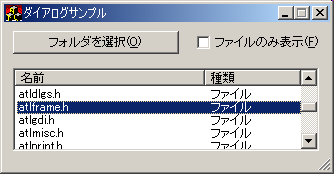
|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CFormView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

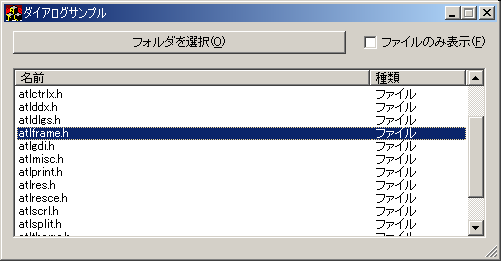
|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "view.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 프로젝트에 폼 용의 다이얼로그 리소스(IDD\_FORMVIEW)를 추가해 리소스 ID를IDD\_FORMVIEW\_FONT 로 변경해 프롭퍼티의 [그 외의 스타일]로 [가시]에 체크를 넣습니다. 그리고 이 다이얼로그 리소스에는 IDC\_LIST\_FONT 이라고 하는 리소스 ID의 리스트 박스 컨트롤을 배치합니다.   
  
  
  
　 stdafx.h 에서는 컨트롤용 클래스를 사용하기 위해서 atlctrls.h 헤더를 인클루드 합니다.   
  
　뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 view.h라고 하는 헤더 파일을 추가해 거기에 CDialogImpl 클래스로부터 파생한 CFormView라고 하는 클래스를 정의하고 있습니다. CFormView 클래스 내에서는 다이얼로그를 작성할 때와 같게 enum 에 의해 다이얼로그 리소스(IDD\_FORMVIEW\_FONT)을 CFormView 클래스에 관련 짓습니다.   
  
WM\_INITDIALOG 메세지 핸들러를 추가해 사용 가능한 TrueType 폰트 명을 열거해 WM\_PAINT 메세지 핸들러를 추가해 m\_strFont에 대입되고 있는 폰트 명을 뷰 윈도우의 하부에 묘화 합니다. 묘화 하는 영역은 GetFontViewRect()이라고 하는 독자적인 함수로 취득합니다. 이 함수는 뷰 윈도우 아래로부터 높이 50의 구형을 돌려줍니다.   
  
WM\_SIZE 메세지 핸들러를 추가해 뷰 윈도우의 상부(뷰 윈도우 아래로부터 높이 50의 구형을 생략한 부분)에 리스트 박스 컨트롤을 배치합니다.   
  
　다음에 통지 코드가 LBN\_SELCHANGE의 커멘드 메세지 핸들러를 추가합니다. 여기에서는 선택되고 있는 폰트 명을 m\_strFont에 대입해 뷰 윈도우의 하부를 갱신합니다.   
  
　이와 같이 폼 뷰 윈도우는 프레임 윈도우의 클라이언트 영역 상에 모달레스 다이아로그를 둔 것 같은 구조가 되어 있습니다. 이 때문에 폼 뷰 윈도우로 발생한 커멘드 메세지(WM\_COMMAND)나 통지 메세지(WM\_NOTIFY) 는 프레임 윈도우는 아니고 직접 뷰 윈도우에 보내집니다.

▣ 다이얼로그 사이즈 > 다이얼로그

**< 다이얼로그 >**  
  
WTL은 CDialogResize 이라고 하는 클래스를 준비해 있습니다. 이 클래스를 다이얼로그 클래스의 base class에 추가하면 다이얼로그의 사이즈를 변경할 때마다 그 자식 컨트롤의 위치나 사이즈도 자동적으로 조정되는 다이얼로그를 작성할 수가 있습니다.   
  
　이하에 나타내는 것은 CDialogResize 클래스를 사용하는 예입니다. 모달 다이얼로그 상의 푸쉬 버튼을 누르면 폴더 선택 다이얼로그가 열려 선택한 폴더 내에 있는 파일명과 폴더명의 일람이 리스트 뷰 컨트롤에 표시됩니다. screen shot은 위가 초기 상태로 아래가 다이얼로그 사이즈를 1.5배에 넓힌 것입니다. 다이얼로그 사이즈를 변경하면 자동적으로 푸쉬 버튼의 폭과 체크 박스의 위치 리스트 뷰 컨트롤의 폭과 높이가 조정됩니다.





|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlframe.h> // CDialogResize클래스를 사용하기 위해(때문에)  #include <atlctrls.h>  #include <atldlgs.h> |

|  |
| --- |
| // maindlg.h  class CMainDlg : public CDialogImpl<CMainDlg>, public CDialogResize<CMainDlg>  {  public:  enum { IDD = IDD\_MAINDLG };  CListViewCtrl m\_list\_file;  CButton m\_check\_fileonly;  // 다이얼로그 리사이즈 맵  BEGIN\_DLGRESIZE\_MAP(CMainDlg)  DLGRESIZE\_CONTROL(IDC\_BUTTON\_OPENDLG, DLSZ\_SIZE\_X)  DLGRESIZE\_CONTROL(IDC\_CHECK\_FILEONLY, DLSZ\_MOVE\_X)  DLGRESIZE\_CONTROL(IDC\_LIST\_DIR, DLSZ\_SIZE\_X | DLSZ\_SIZE\_Y)  END\_DLGRESIZE\_MAP()  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMainDlg)  MSG\_WM\_INITDIALOG(OnInitDialog)  COMMAND\_ID\_HANDLER\_EX(IDC\_BUTTON\_OPENDLG, OnButtonOpenDlg)  COMMAND\_ID\_HANDLER\_EX(IDOK, OnOK)  COMMAND\_ID\_HANDLER\_EX(IDCANCEL, OnCancel)  CHAIN\_MSG\_MAP(CDialogResize<CMainDlg>) // CDialogResize클래스에의 체인  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  // 스크린의 중앙에 배치  CenterWindow();  // 큰 아이콘 설정  HICON hIcon = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXICON), ::GetSystemMetrics(SM\_CYICON));  SetIcon(hIcon, TRUE);    // 작은 아이콘 설정  HICON hIconSmall = AtlLoadIconImage(IDR\_MAINFRAME, LR\_DEFAULTCOLOR,  ::GetSystemMetrics(SM\_CXSMICON), ::GetSystemMetrics(SM\_CYSMICON));  SetIcon(hIconSmall, FALSE);  // 컨트롤 설정  m\_check\_fileonly = GetDlgItem(IDC\_CHECK\_FILEONLY);  m\_list\_file = GetDlgItem(IDC\_LIST\_DIR);  m\_list\_file.SetExtendedListViewStyle(LVS\_EX\_INFOTIP | LVS\_EX\_FULLROWSELECT);  // 리스트뷰콘트로르에 컬럼 삽입  CRect rcList;  m\_list\_file.GetWindowRect(rcList);  int nScrollWidth = GetSystemMetrics(SM\_CXVSCROLL);  int n3DEdge = GetSystemMetrics(SM\_CXEDGE);  m\_list\_file.InsertColumn(0, \_T("이름"), LVCFMT\_LEFT, 190, -1);  m\_list\_file.InsertColumn(1, \_T("종류"), LVCFMT\_LEFT,  rcList.Width() - 190 - nScrollWidth - n3DEdge \* 2, -1);  // 다이얼로그 리사이즈 초기화  DlgResize\_Init(true, true, WS\_THICKFRAME | WS\_MAXIMIZEBOX | WS\_CLIPCHILDREN);  return TRUE;  }  void OnButtonOpenDlg(UINT uNotifyCode, int nID, HWND hWndCtl){  CFolderDialog dlg(NULL, \_T("폴더를 선택해 주세요. "));  if(dlg.DoModal() == IDOK){  m\_list\_file.DeleteAllItems();  // 리스트뷰콘트로르에 아이템 추가  CString strFind;  strFind.Format(\_T("%s\\\*"), dlg.GetFolderPath());  CFindFile find;  if(find.FindFile(strFind)){  do{  if(find.IsDots())  continue;  if(m\_check\_fileonly.GetCheck() && find.IsDirectory())  continue;  int nIndex = m\_list\_file.GetItemCount();  m\_list\_file.AddItem(nIndex, 0, find.GetFileName());  m\_list\_file.AddItem(nIndex, 1,  find.IsDirectory() ? \_T("폴더") : \_T("파일"));  }while(find.FindNextFile());  }  }  }  void OnOK(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  void OnCancel(UINT uNotifyCode, int nID, HWND hWndCtl){  EndDialog(nID);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "maindlg.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMainDlg dlgMain;  nRet = dlgMain.DoModal();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 리소스를 작성합니다. 다이얼로그에 버튼 컨트롤과 리스트 뷰 컨트롤을 배치해 리소스 ID를 다음과 같이 지정합니다. 덧붙여 리스트 뷰 컨트롤의 [스타일]에서는 디폴트에 가세해 [표시]로 [리포트]를 선택합니다.

|  |  |
| --- | --- |
| 컨트롤명 | 리소스 ID |
| 푸쉬 버튼 | IDC\_BUTTON\_OPENDLG |
| 체크 박스 | IDC\_CHECK\_FILEONLY |
| 리스트뷰 | IDC\_LIST\_DIR |

　다음에 stdafx.h 에서는 CDialogResize 클래스를 사용하기 위해서 atlframe.h 헤더를 인클루드 합니다.   
  
　CMainDlg 클래스에서는 우선 base class에 CDialogResize 클래스를 추가해 메세지 맵에 CDialogResize 클래스에의 체인을 추가합니다. CDialogResize 클래스는 내부에 메세지 맵을 준비하고 있어 다이얼로그 사이즈가 변경되어 WM\_SIZE 메세지가 보내져 왔을 때에 다이얼로그상의 컨트롤의 위치나 사이즈를 조정합니다.   
  
　다음에 다이얼로그 리사이즈 맵을 준비합니다. 이 맵에 전용의 매크로로 엔트리를 추가하는 것으로 특정의 컨트롤의 사이즈 또는 위치 또는 양쪽 모두를 변경시킬 수가 있습니다. 엔트리용 매크로는 다음과 같이 되어 있습니다.

* DLGRESIZE\_CONTROL(컨트롤ID, 플래그)

플래그에는 다음과 같은 종류가 있습니다.

* DLSZ\_SIZE\_X

다이얼로그의 폭이 변경되었을 때 동시에 컨트롤의 폭을 변경합니다.

* DLSZ\_SIZE\_Y

다이얼로그의 높이가 변경되었을 때 동시에 컨트롤의 높이를 변경합니다.

* DLSZ\_MOVE\_X

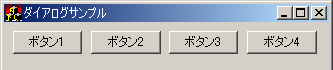
다이얼로그의 폭이 변경되었을 때 동시에 컨트롤의 위치를 수평 방향으로 이동합니다.

* DLSZ\_MOVE\_Y

다이얼로그의 높이가 변경되었을 때 동시에 컨트롤의 위치를 수직 방향으로 이동합니다.

* DLSZ\_REPAINT

컨트롤의 사이즈나 위치가 변경되었을 때 컨트롤을 재 묘화 합니다.

엔트리용 매크로에는 컨트롤을 그룹화 하는 매크로가 준비되어 있습니다. 예를 들면 다이얼로그에 다음과 같이 버튼이 배치되고 있었다고 합니다.   
  
  
  
각각의 버튼에는IDC\_BUTTON1~IDC\_BUTTON4라고 하는ID를 설정합니다. 다음에 다이얼로그 리 사이즈 매크로에 다음과 같은 엔트리를 추가합니다.

|  |
| --- |
| BEGIN\_DLGRESIZE\_MAP(CMainDlg)  // 4개의 버튼을 그룹화  BEGIN\_DLGRESIZE\_GROUP()  DLGRESIZE\_CONTROL(IDC\_BUTTON1, DLSZ\_SIZE\_X)  DLGRESIZE\_CONTROL(IDC\_BUTTON2, DLSZ\_SIZE\_X)  DLGRESIZE\_CONTROL(IDC\_BUTTON3, DLSZ\_SIZE\_X)  DLGRESIZE\_CONTROL(IDC\_BUTTON4, DLSZ\_SIZE\_X)  END\_DLGRESIZE\_GROUP()  END\_DLGRESIZE\_MAP() |

이렇게 하면 다이얼로그의 사이즈를 변경했을 때에 그룹화 된 각 버튼의 사이즈와 위치가 자동적으로 균등하게 배치되게 됩니다.   
  
  
  
　다음에 WM\_INITDIALOG 메세지 핸들러로 DlgResize\_Init()를 호출합니다. 제1인수는 다이얼로그의 우하로 사이즈 변경용의 그립을 표시하는지 어떤지를 나타내는bool값, 제2인수는 다이얼로그를 변경할 수 있는 최소 사이즈를 다이얼로그 자원의 사이즈에 고정하는지 어떤지를 나타내는bool값, 제3인수는 다이얼로그에 추가하는 윈도우 스타일을 지정합니다. 리 사이즈 가능한 다이얼로그를 작성하는 경우는 WS\_THICKFRAME 스타일을 추가할 필요가 있습니다. 모든 인수는 생략 가능합니다. DlgResize\_Init()의 정의에서는 각 인수의 디폴트 인수는 다음과 같이 되어 있습니다.

|  |
| --- |
| void DlgResize\_Init(bool bAddGripper = true,  bool bUseMinTrackSize = true, DWORD dwForceStyle = WS\_CLIPCHILDREN)  {  ...  ... |

　마지막으로 리소스 ID가 IDC\_BUTTON\_OPENDLG의 커멘드 메세지 핸들러를 추가해 폴더 선택 다이얼로그를 표시해 선택한 폴더내의 파일명과 폴더명의 일람을 리스트 뷰 컨트롤에 표시합니다. [파일만 표시] 체크 박스에 체크가 들어가 있는 경우는 파일명만 표시합니다.

▣ 다이얼로그 사이즈 > 폼

**< 폼 >**  
  
CDialogResize 클래스는 폼 뷰에서도 사용할 수 있습니다. 이하에 나타내는 것은 폼 뷰 윈도우의 base class로서 CDialogResize 클래스를 사용하는 예입니다. 뷰 윈도우의 상부에는 사용 가능한 TrueType폰트를 열거한 리스트 박스 컨트롤을 배치해 폰트를 선택하면 뷰 윈도우의 하부(뷰 윈도우 아래로부터 높이50의 구형) 에 그 폰트로 폰트명을 묘화 합니다.

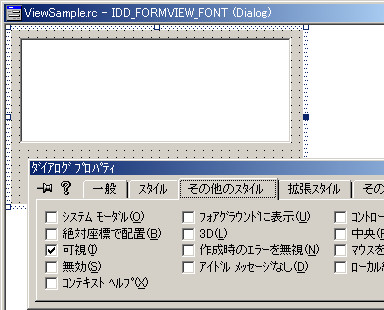


|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

|  |
| --- |
| // view.h  class CResizeFormView : public CDialogImpl<CResizeFormView>,  public CDialogResize<CResizeFormView>  {  public:  enum { IDD = IDD\_FORMVIEW\_FONT };  BOOL PreTranslateMessage(MSG\* pMsg){  return CWindow::IsDialogMessage(pMsg);  }  BEGIN\_DLGRESIZE\_MAP(CResizeFormView)  DLGRESIZE\_CONTROL(IDC\_LIST\_FONT, DLSZ\_SIZE\_X | DLSZ\_SIZE\_Y)  END\_DLGRESIZE\_MAP()  private:  CListBox m\_list\_font;  CString m\_strFont; // 묘화 하는 폰트명  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CResizeFormView)  MSG\_WM\_INITDIALOG(OnInitDialog)  MSG\_WM\_PAINT(OnPaint)  COMMAND\_CODE\_HANDLER\_EX(LBN\_SELCHANGE, OnListSelChange)  CHAIN\_MSG\_MAP(CDialogResize<CResizeFormView>)  END\_MSG\_MAP()  LRESULT OnInitDialog(HWND hWnd, LPARAM lParam){  m\_list\_font = GetDlgItem(IDC\_LIST\_FONT);  // 리스트 박스 컨트롤을 배치  CRect rcForm;  GetClientRect(rcForm);  rcForm.bottom -= 50;  m\_list\_font.MoveWindow(rcForm);  // 다이얼로그 리사이즈 초기화  DlgResize\_Init(false, false);  // 사용 가능한TrueType폰트명을 열거  CClientDC dc(m\_hWnd);  EnumFontFamilies(dc, NULL,  (FONTENUMPROC)FontProc, (LPARAM)m\_list\_font.m\_hWnd);  return TRUE;  }  static int CALLBACK FontProc(ENUMLOGFONT \*lpelf,  NEWTEXTMETRIC \*lpntm, int nFontType, LPARAM lParam)  {  CListBox list = (HWND)lParam;  // TrueType폰트명만 리스트에 추가  if(nFontType & TRUETYPE\_FONTTYPE)  list.AddString(lpelf->elfLogFont.lfFaceName);  return 1;  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  if(!m\_strFont.IsEmpty()){  // 사이즈20포인트의 폰트를 작성해 , 선택  CFont font;  font.CreatePointFont(200, m\_strFont);  HFONT hOldFont = dc.SelectFont(font);  // 뷰윈드우 하부에 폰트명을 묘화  CRect rcFontView = GetFontViewRect();  dc.SetBkMode(TRANSPARENT);  dc.DrawText(m\_strFont,  m\_strFont.GetLength(), rcFontView, DT\_VCENTER | DT\_SINGLELINE);  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  }  }  void OnListSelChange(UINT uNotifyCode, int nID, HWND hWndCtl){  // 현재 선택되고 있는 아이템을 취득해 표시  int nIndex = m\_list\_font.GetCurSel();  if(nIndex != LB\_ERR){  m\_list\_font.GetText(nIndex, m\_strFont);  // 뷰윈드우 하부를 갱신  InvalidateRect(GetFontViewRect());  }  }  CRect GetFontViewRect(){  // 폰트명을 묘화 하는 구형(뷰윈드우아래로부터 높이50의 구형) 을 취득  CRect rcForm;  GetClientRect(rcForm);  rcForm.top = rcForm.bottom - 50;  return rcForm;  }  }; |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CResizeFormView m\_view;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_CLIENT\_COMMANDS() // 뷰크라스에 커멘드 체인  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 뷰를 작성  m\_hWndClient = m\_view.Create(m\_hWnd);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "view.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 프로젝트에 폼용의 다이얼로그 리소스(IDD\_FORMVIEW)를 추가해 리소스 ID를IDD\_FORMVIEW\_FONT로 변경해 프롭퍼티의 [그 외의 스타일]로 [가시]에 체크를 넣습니다. 그리고 이 다이얼로그 리소스에는 IDC\_LIST\_FONT 이라고 하는 리소스 ID의 리스트 박스 컨트롤을 배치합니다.   
  
  
  
　다음에 뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 view.h라고 하는 헤더 파일을 추가해 거기에 CDialogImpl 클래스와 CDialogResize 클래스로부터 파생한 CResizeFormView 라고 하는 클래스를 정의하고 있습니다. 여기에서는 다이얼로그를 작성할 때와 같게 enum에 의해 다이얼로그 리소스(IDD\_FORMVIEW\_FONT)를 CResizeFormView 클래스에 관련 짓습니다.   
  
　다음에 다이얼로그 리 사이즈 맵에 리스트 박스 컨트롤을 위한 엔트리를 추가해 메세지 맵에는 CDialogResize 클래스에의 체인을 추가합니다.   
  
　다음에 WM\_INITDIALOG 메세지 핸들러를 추가해 리스트 박스 컨트롤을 뷰 윈도우 아래로부터 높이 50를 뺀 구형에 배치해 DlgResize\_Init()를 호출합니다. 위의 예에서는 상태 바를 사용하고 있어 벌써 사이즈 변경 용의 그립이 표시되고 있으므로 DlgResize\_Init()의 제1인수에는 false을 지정합니다. 그 후 사용 가능한 TrueType 폰트 명을 열거합니다.   
  
　다음에 WM\_PAINT 메세지 핸들러를 추가해 m\_strFont에 대입되고 있는 폰트 명을 뷰 윈도우의 하부에 묘화 합니다. 묘화 하는 영역은 GetFontViewRect() 이라고 하는 독자적인 함수로 취득합니다. 이 함수는 뷰 윈도우 아래로부터 높이 50의 구형을 돌려줍니다.   
  
　다음에 통지 코드가 LBN\_SELCHANGE의 커멘드 메세지 핸들러를 추가합니다. 여기에서는 선택되고 있는 폰트 명을 m\_strFont에 대입해 뷰 윈도우의 하부를 갱신합니다.   
  
　이와 같이 폼뷰 윈도우에 CDialogResize클래스를 사용하면 스스로 WM\_SIZE 메세지 핸들러를 추가해 윈도우 사이즈가 변경될 때마다 폼 상의 컨트롤을 재배치할 필요가 없어집니다.

▣ 다이얼로그 사이즈 > 윈도우

**< 윈도우 >**  
  
CDialogResize 클래스는 (폼을 포함한다 ) 다이얼로그 뿐만이 아니라 통상의 윈도우에서도 사용할 수 있습니다. 이하에 나타내는 것은 프레임 윈도우의 base class로서 CDialogResize클래스를 사용하는 예입니다. 윈도우의 상부에는 사용 가능한 TrueType 폰트를 열거한 리스트 박스 컨트롤을 배치해 폰트를 선택하면 윈도우의 하부(윈도우아래로부터 높이50의 구형)에 그 폰트로 폰트 명을 묘화 합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h> |

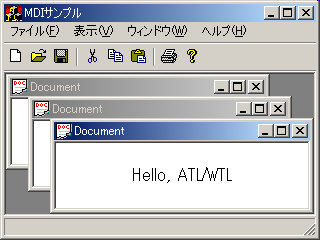
|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>, public CUpdateUI<CMainFrame>,  public CMessageFilter, public CIdleHandler, public CDialogResize<CMainFrame>  {  public:  DECLARE\_FRAME\_WND\_CLASS\_EX(NULL, IDR\_MAINFRAME, 0, COLOR\_BTNFACE)  enum { IDC\_LIST\_FONT = 1001 };  CListBox m\_list\_font;  CString m\_strFont; // 묘화 하는 폰트명  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_DLGRESIZE\_MAP(CMainFrame)  DLGRESIZE\_CONTROL(IDC\_LIST\_FONT, DLSZ\_SIZE\_X | DLSZ\_SIZE\_Y)  END\_DLGRESIZE\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_PAINT(OnPaint)  COMMAND\_HANDLER\_EX(IDC\_LIST\_FONT, LBN\_SELCHANGE, OnListSelChange)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnMenuExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  CHAIN\_MSG\_MAP(CDialogResize<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 리스트 박스 컨트롤을 배치  CRect rcList;  GetClientRect(rcList);  CRect rcStatusbar;  ::GetClientRect(m\_hWndStatusBar, rcStatusbar);  rcList.bottom -= (50 + rcStatusbar.Height());  m\_list\_font.Create(m\_hWnd, rcList, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN | WS\_VSCROLL |  LBS\_NOINTEGRALHEIGHT | LBS\_NOTIFY | LBS\_WANTKEYBOARDINPUT | LBS\_SORT,  WS\_EX\_CLIENTEDGE, IDC\_LIST\_FONT);  m\_list\_font.SetFont(AtlGetDefaultGuiFont());  // 다이얼로그 리사이즈 초기화  DlgResize\_Init(false, false);  // 사용 가능한TrueType폰트명을 열거  CClientDC dc(m\_hWnd);  EnumFontFamilies(dc, NULL,  (FONTENUMPROC)FontProc, (LPARAM)m\_list\_font.m\_hWnd);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  static int CALLBACK FontProc(ENUMLOGFONT \*lpelf,  NEWTEXTMETRIC \*lpntm, int nFontType, LPARAM lParam)  {  CListBox list = (HWND)lParam;  // TrueType폰트명만 리스트에 추가  if(nFontType & TRUETYPE\_FONTTYPE)  list.AddString(lpelf->elfLogFont.lfFaceName);  return 1;  }  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  if(!m\_strFont.IsEmpty()){  // 사이즈20포인트의 폰트를 작성해 , 선택  CFont font;  font.CreatePointFont(200, m\_strFont);  HFONT hOldFont = dc.SelectFont(font);  // 윈도우 하부에 폰트명을 묘화  CRect rcFontView = GetFontViewRect();  dc.SetBkMode(TRANSPARENT);  dc.DrawText(m\_strFont,  m\_strFont.GetLength(), rcFontView, DT\_VCENTER | DT\_SINGLELINE);  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  }  }  void OnListSelChange(UINT uNotifyCode, int nID, HWND hWndCtl){  // 현재 선택되고 있는 아이템을 취득해 표시  int nIndex = m\_list\_font.GetCurSel();  if(nIndex != LB\_ERR){  m\_list\_font.GetText(nIndex, m\_strFont);  // 윈도우 하부를 갱신  InvalidateRect(GetFontViewRect());  }  }  CRect GetFontViewRect(){  // 폰트명을 묘화 하는 구형(윈도우아래로부터 높이50의 구형) 을 취득  CRect rcForm;  GetClientRect(rcForm);  CRect rcStatusbar;  ::GetClientRect(m\_hWndStatusBar, rcStatusbar);  rcForm.top = rcForm.bottom - (50 + rcStatusbar.Height());  rcForm.bottom -= rcStatusbar.Height();  return rcForm;  }  void OnMenuExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "mainfrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 프레임 윈도우인 CMainFrame 클래스의 base class에 CDialogResize 클래스를 추가합니다.   
  
　CMainFrame 클래스 내에서는 우선 윈도우의 폰트 명을 묘화 하는 영역을 다이얼로그의 색과 같게 하기 위한 DECLARE\_FRAME\_WND\_CLASS\_EX 매크로를 사용해 제4인수에COLOR\_BTNFACE를 지정합니다.   
  
　다음에 WM\_CREATE 메세지 핸들러로 리스트 박스 컨트롤을 작성합니다. 이 때 리스트 박스 컨트롤은 상태 바의 높이를 고려해 배치합니다. 그 후 DlgResize\_Init()를 호출합니다. 위의 예에서는 상태 바를 사용하고 있어 벌써 사이즈 변경용의 그립이 표시되고 있으므로 , DlgResize\_Init()의 제1인수에는 false을 지정합니다.   
  
　다음에 다이얼로그 리 사이즈 맵에 리스트 박스 컨트롤을 위한 엔트리를 추가해 메세지 맵에는 CDialogResize 클래스에의 체인을 추가합니다.   
  
　다음에 WM\_PAINT 메세지 핸들러를 추가해 m\_strFont에 대입되고 있는 폰트 명을 윈도우의 하부에 묘화 합니다. 묘화 하는 영역은 GetFontViewRect() 이라고 하는 독자적인 함수로 취득합니다. 이 함수는 윈도우 아래로부터 상태 바의 높이를 고려해 높이 50의 구형을 돌려줍니다.   
  
　다음에 통지 코드가 LBN\_SELCHANGE의 커멘드 메세지 핸들러를 추가합니다. 여기에서는 선택되고 있는 폰트명을 m\_strFont에 대입해 윈도우의 하부를 갱신합니다.   
  
　이와 같이 CDialogResize 클래스는 다이얼로그와 같이 윈도우라도 사용할 수 있으므로 스스로 WM\_SIZE 메세지 핸들러를 추가해 윈도우 사이즈가 변경될 때마다 컨트롤을 재배치할 필요가 없어집니다. 그리고 DlgResize\_Init()의 제2인수에 true를 지정하면 윈도우 사이즈를 초기 사이즈보다 작게 변경할 수 없는 것 같은 윈도우를 작성할 수가 있습니다.

▣ MDI > MDI 윈도우

**< MDI 윈도우 >**  
  
WTL은 MDI(Multiple Document Interface) 어플리케이션을 작성하기 위한 클래스를 준비해 있습니다. MDI 프레임 윈도우를 작성하기 위해서는 CMDIFrameWindowImpl클래스를 그 클라이언트 영역에 표시되는 아이 윈도우를 작성하기 위해서는CMDIChildWindowImpl클래스를 사용합니다. 이것들은 CWindowImpl 클래스와 같이 base class로서의 보고 사용합니다.   
  
　이하에 나타내는 것은 WTL의 MDI 서포트 클래스를 사용해 MDI 어플리케이션을 작성하는 예입니다. 각 MDI 자식 윈도우의 뷰 의 중앙에는 「Hello, ATL/WTL」이라고 하는 캐릭터 라인을 표시합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #define \_WTL\_MDIWINDOWMENU\_TEXT \_T("윈도우(&W)")  #include <atlframe.h> |

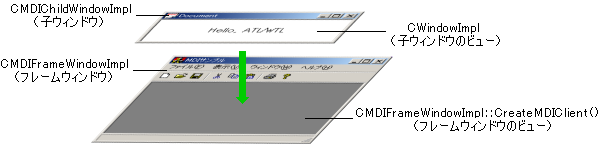
|  |
| --- |
| // MDIview.h  class CMDIView : public CWindowImpl<CMDIView>  {  public:  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMDIView)  MSG\_WM\_PAINT(OnPaint)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  }; |

|  |
| --- |
| // ChildFrm.h  class CChildFrame : public CMDIChildWindowImpl<CChildFrame>  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MDICHILD)  CMDIView m\_view;  virtual void OnFinalMessage(HWND /\*hWnd\*/){  delete this;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CChildFrame)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_FORWARDMSG(OnForwardMsg)  CHAIN\_MSG\_MAP(CMDIChildWindowImpl<CChildFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 뷰윈드우를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN,  WS\_EX\_CLIENTEDGE);  SetMsgHandled(false);  return 1;  }  LRESULT OnForwardMsg(LPMSG pMsg, DWORD dwUserData){  // 메세지 필터 처리  if(CMDIChildWindowImpl<CChildFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  }; |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CMDIFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CMDIFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  // 액티브한 아이 윈도우에 메세지 필터의 기회를 주는  HWND hWnd = MDIGetActive();  if(hWnd != NULL)  return (BOOL)::SendMessage(hWnd, WM\_FORWARDMSG, 0, (LPARAM)pMsg);  return FALSE;  }  virtual BOOL OnIdle(){  UIUpdateToolBar();  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnFileExit)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_NEW, OnFileNew)  COMMAND\_ID\_HANDLER\_EX(ID\_WINDOW\_CASCADE, OnWindowCascade)  COMMAND\_ID\_HANDLER\_EX(ID\_WINDOW\_TILE\_HORZ, OnWindowTile)  COMMAND\_ID\_HANDLER\_EX(ID\_WINDOW\_ARRANGE, OnWindowArrangeIcons)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CMDIFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 툴바를 작성  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  UIAddToolBar(hWndToolBar);  // 리버를 작성해 툴바를 밴드에 추가  CreateSimpleReBar();  AddSimpleReBarBand(hWndToolBar);  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 프레임 윈도우의 뷰윈드우를 작성  CreateMDIClient();  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnFileExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  void OnFileNew(UINT uNotifyCode, int nID, HWND hWndCtl){  // 자 윈도우를 작성  CChildFrame\* pChild = new CChildFrame;  pChild->CreateEx(m\_hWndClient);  }  void OnWindowCascade(UINT uNotifyCode, int nID, HWND hWndCtl){  // 자 윈도우를 거듭해 표시  MDICascade();  }  void OnWindowTile(UINT uNotifyCode, int nID, HWND hWndCtl){  // 자 윈도우를 상하에 늘어놓아 표시  MDITile();  }  void OnWindowArrangeIcons(UINT uNotifyCode, int nID, HWND hWndCtl){  // 최소화된 아이 윈도우의 아이콘을 정렬  MDIIconArrange();  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "MDIView.h"  #include "ChildFrm.h"  #include "MainFrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

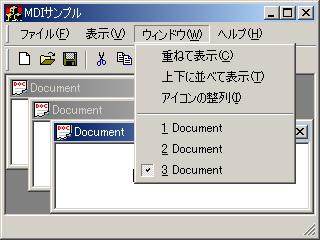
　우선 stdafx.h 에서는 MDI 서포트 클래스를 사용하기 위해서 atlframe.h 헤더를 인클루드 합니다만 그 전에 \_WTL\_MDIWINDOWMENU\_TEXT를 정의합니다. 이것은 자식 윈도우용 도구모음 자원(MDI에서는 자식 윈도우가 액티브하게 되면 도구모음이 자식 윈도우용의 것에 바뀝니다)의 [윈도우] 메뉴 아이템 캐릭터 라인입니다. MDI어플리케이션에서는 통상 자식 윈도우가 열리고 있는 경우는 [윈도우] 메뉴 아이템 이하에 자식 윈도우의 캡션 리스트를 추가합니다.   
  
  
  
WTL이 캡션 리스트를 추가하기 위한 메뉴 아이템의 위치를 확인하기 위해서 \_WTL\_MDIWINDOWMENU\_TEXT 그리고 정의된 캐릭터 라인을 사용합니다. 덧붙여 정의하지 않으면 디폴트로 "&Window" 라고 하는 캐릭터 라인이 정의됩니다. 그리고 [윈도우] 메뉴 아이템은 도구모음의 오른쪽으로부터 두 번째에 없으면 안됩니다. WTL은 도구모음의 오른쪽으로부터 두 번째의 아이템이 \_WTL\_MDIWINDOWMENU\_TEXT로 정의된 캐릭터 라인이었을 경우에만 그 아이템 이하에 캡션 리스트를 추가합니다.   
  
　다음에 MDI의 각 윈도우용 클래스를 정의합니다. WTL MDI 어플리케이션의 프레임 윈도우는 CMDIFrameWindowImpl클래스의 상속 클래스 자식 윈도우는 CMDIChildWindowImpl 클래스의 상속 클래스 자식 윈도우의 뷰 윈도우는 통상의 뷰 윈도우와 같은 CWindowImpl클래스의 상속 클래스를 사용합니다. 덧붙여 프레임 윈도우의 뷰 윈도우(윈도우 클래스가 "MDIClient"라고 하는MDI전용의 특별한 윈도우)는 CMDIFrameWindowImpl클래스의 멤버 함수인 CreateMDIClient()을 호출해 작성합니다. 다음에 나타내는 것은 WTL에 의한 MDI 어플리케이션의 이미지도 입니다.



　우선 자식 윈도우의 뷰 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에MDIview.h 라고 하는 헤더 파일을 추가해 거기에 CWindowImpl 클래스로부터 파생한CMDIView 라고 하는 클래스를 정의하고 있습니다. 이 클래스는 WM\_PAINT 메세지 핸들러를 추가해 뷰의 중앙에 캐릭터 라인을 표시하고 있을 뿐입니다.   
  
　다음에 자식 윈도우를 위한 클래스를 정의합니다. 위의 예에서는 프로젝트에 ChildFrm.h라고 하는 헤더 파일을 추가해 거기에 CMDIChildWindowImpl 클래스로부터 파생한CChildFrame 라고 하는 클래스를 정의하고 있습니다. CMDIChildWindowImpl 클래스는CFrameWindowImpl 클래스와 같이 CFrameWindowImplBase 라고 하는 클래스로부터 파생하고 있어 기본적인 프레임 윈도우의 기능을 가지고 있습니다.   
  
CChildFrame 클래스는 WM\_CREATE 메세지 핸들러로 뷰 윈도우를 작성해 WM\_FORWARDMSG 메세지 핸들러로 메세지 필터 처리를 실시합니다. 그리고 자식 윈도우는MDI 프레임 윈도우로부터 동적으로 작성되기 때문에 자식 윈도우 파기의 타이밍 즉OnFinalMessage()으로 자기 자신을 삭제합니다.   
  
　다음에 MDI 프레임 윈도우를 위한 클래스를 정의합니다. MDI 프레임 윈도우는CMDIFrameWindowImpl 클래스로부터 파생하고 있는 것 외는 거의 통상의 프레임 윈도우와 같은 구조입니다. CMDIFrameWindowImpl 클래스는 CFrameWindowImpl 클래스와 같이 CFrameWindowImplBase 라고 하는 클래스로부터 파생하고 있어 기본적인 프레임 윈도우의 기능을 가지고 있습니다.   
  
CMainFrame 클래스는 WM\_CREATE 메세지 핸들러로 각 바를 작성해 CreateMDIClient() 를 호출해 프레임 윈도우 용의 뷰 윈도우를 작성합니다. 이 함수의 제1인수에는 자식 윈도우의 캡션 리스트를 추가하는 메뉴 핸들, 제2인수에는 뷰 윈도우를 식별하는ID, 제3인수에는 최초로 표시되는 아이 윈도우의 메뉴ID를 지정할 수 있습니다만 모두 생략 가능합니다.   
  
　다음에 커멘드 메세지 핸들러를 추가합니다. ID하지만ID\_FILE\_NEW의 커멘드 메세지 핸들러에서는 아이 윈도우를 동적으로 작성합니다.   
  
덧붙여atlframe.h헤더에는 ,MDI프레임 윈도우에 보내진 커멘드 메세지를 액티브한 아이 윈도우에 전송 하기 위해서 , CHAIN\_MDI\_CHILD\_COMMANDS()라고 하는 커멘드 체인 매크로가 정의되고 있습니다.

▣ MDI > MDI 커맨드 바

**< MDI 커맨드 바 >**  
  
WTL은 MDI 어플리케이션용의 커맨드 바를 작성하기 위해서 CMDICommandBarCtrl라고 하는 클래스를 준비해 있습니다. 이 클래스는 통상의 커맨드 바와 동등의 기능을 가져 사용법도 통상의 커맨드 바와 거의 같습니다.   
  
　이하에 나타내는 것은 CMDICommandBarCtrl을 사용하는 예입니다. 전회 작성한 MDI 어플리케이션의 도구모음을 MDI용 커맨드 바로 변경합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlctrlw.h>  #define \_WTL\_MDIWINDOWMENU\_TEXT \_T("윈도우(&W)")  #include <atlframe.h> |

|  |
| --- |
| // MDIview.h  class CMDIView : public CWindowImpl<CMDIView>  {  public:  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CMDIView)  MSG\_WM\_PAINT(OnPaint)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  CRect rect;  GetClientRect(rect);  dc.DrawText(\_T("Hello, ATL/WTL"), -1,  rect, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);  }  }; |

|  |
| --- |
| // ChildFrm.h  class CChildFrame : public CMDIChildWindowImpl<CChildFrame>  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MDICHILD)  CMDIView m\_view;  virtual void OnFinalMessage(HWND /\*hWnd\*/){  delete this;  }  private:  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CChildFrame)  MSG\_WM\_CREATE(OnCreate)  MSG\_WM\_FORWARDMSG(OnForwardMsg)  CHAIN\_MSG\_MAP(CMDIChildWindowImpl<CChildFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 뷰윈드우를 작성  m\_hWndClient = m\_view.Create(m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN,  WS\_EX\_CLIENTEDGE);  SetMsgHandled(false);  return 1;  }  LRESULT OnForwardMsg(LPMSG pMsg, DWORD dwUserData){  // 메세지 필터 처리  if(CMDIChildWindowImpl<CChildFrame>::PreTranslateMessage(pMsg))  return TRUE;  return m\_view.PreTranslateMessage(pMsg);  }  }; |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CMDIFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CMDICommandBarCtrl m\_CmdBar;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CMDIFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  // 액티브한 아이 윈도우에 메세지 필터의 기회를 주는  HWND hWnd = MDIGetActive();  if(hWnd != NULL)  return (BOOL)::SendMessage(hWnd, WM\_FORWARDMSG, 0, (LPARAM)pMsg);  return FALSE;  }  virtual BOOL OnIdle(){  UIUpdateToolBar();  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnFileExit)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_NEW, OnFileNew)  COMMAND\_ID\_HANDLER\_EX(ID\_WINDOW\_CASCADE, OnWindowCascade)  COMMAND\_ID\_HANDLER\_EX(ID\_WINDOW\_TILE\_HORZ, OnWindowTile)  COMMAND\_ID\_HANDLER\_EX(ID\_WINDOW\_ARRANGE, OnWindowArrangeIcons)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CMDIFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 코맨드 바를 작성  HWND hWndCmdBar = m\_CmdBar.Create(m\_hWnd,  rcDefault, NULL, ATL\_SIMPLE\_CMDBAR\_PANE\_STYLE);  // 기존의 도구모음을 아탓치  m\_CmdBar.AttachMenu(GetMenu());  // 메뉴 아이템에 관련지을 수 있었던 툴바의 화상을 로드  m\_CmdBar.LoadImages(IDR\_MAINFRAME);  // 기존의 도구모음을 삭제  SetMenu(NULL);  // 툴바를 작성  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  UIAddToolBar(hWndToolBar);  // 리버를 작성해 , 코맨드 바와 툴바를 밴드에 추가  CreateSimpleReBar(ATL\_SIMPLE\_REBAR\_NOBORDER\_STYLE);  AddSimpleReBarBand(hWndCmdBar);  AddSimpleReBarBand(hWndToolBar, NULL, TRUE);  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 프레임 윈도우의 뷰윈드우를 작성  CreateMDIClient();  m\_CmdBar.SetMDIClient(m\_hWndMDIClient);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnFileExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  void OnFileNew(UINT uNotifyCode, int nID, HWND hWndCtl){  // 자 윈도우를 작성  CChildFrame\* pChild = new CChildFrame;  pChild->CreateEx(m\_hWndClient);  }  void OnWindowCascade(UINT uNotifyCode, int nID, HWND hWndCtl){  // 자 윈도우를 거듭해 표시  MDICascade();  }  void OnWindowTile(UINT uNotifyCode, int nID, HWND hWndCtl){  // 자 윈도우를 상하에 늘어놓아 표시  MDITile();  }  void OnWindowArrangeIcons(UINT uNotifyCode, int nID, HWND hWndCtl){  // 최소화된 아이 윈도우의 아이콘을 정렬  MDIIconArrange();  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "MDIView.h"  #include "ChildFrm.h"  #include "MainFrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

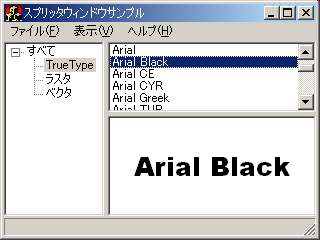
　우선 stdafx.h 에서는 CMDICommandBarCtrl 클래스를 사용하기 위해서 atlctrlw.h 헤더를 인클루드 합니다.   
  
　다음으로 변경하는 것은 CMainFrame 클래스입니다. CMainFrame 클래스의 멤버 변수로서 CMDICommandBarCtrl 클래스의 인스턴스를 선언해 WM\_CREATE 메세지 핸들러로Create() 를 호출하는 것에 의해 MDI 커맨드 바를 작성합니다.   
  
커맨드 바를 작성한 후 는 커맨드 바에 메뉴 아이템을 추가하거나 메뉴 아이템에 관련 짓는 화상을 로드 합니다만 이것들은 통상의 커맨드 바의 경우와 같습니다.   
  
　마지막으로 CreateMDIClient()를 호출한 후 에 CMDICommandBarCtrl 클래스의 멤버 함수인 SetMDIClient()의 인수에 MDI프레임 윈도우의 뷰 윈도우를 지정해 호출합니다. 이 핸들을 보관 유지하는 m\_hWndMDIClient은 CMDIFrameWindowImpl 클래스의 멤버 변수로 CreateMDIClient()를 호출하면 설정 됩니다.

▣ 스플릿터 윈도우 >

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **< 스플리터 윈도우 >**  WTL은 스플리터 윈도우를 CSplitterWindowT 이라고 하는 템플릿 클래스에서 캡슐화하고 있습니다. 템플릿 인수는 bool 값으로 true의 경우는 윈도우를 수직 방향으로 분할해 false의 경우는 수평 방향으로 분할합니다. atlsplit.h 헤더에서는 typedef에 의해 다음과 같이 선언되고 있습니다.   |  | | --- | | // atlsplit.h  typedef CSplitterWindowT<true> CSplitterWindow;  typedef CSplitterWindowT<false> CHorSplitterWindow; |   　이하에 나타내는 것은 CSplitterWindow 클래스를 사용해 스플리터 윈도우를 작성하는 예입니다. 왼쪽 페인에는 사용 가능한 폰트명을 리스트에 표시해 오른쪽 페인에는 리스트로 선택된 폰트로 폰트명을 표시합니다. 메뉴의[표시]-[뷰 변경]을 실행하면 오른쪽 페인만을 표시합니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl20-01/01.png  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl20-01/02.png   |  | | --- | | // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h>  #include <atlsplit.h> // 스플리터 윈도우를 사용하기 위해(때문에) |  |  | | --- | | // LeftView.h  class CLeftView : public CWindowImpl<CLeftView, CListBox>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CListBox::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CLeftView)  MSG\_WM\_CREATE(OnCreate)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetFont(AtlGetDefaultGuiFont());  CClientDC dc(m\_hWnd);  EnumFontFamilies(dc, NULL,  (FONTENUMPROC)FontProc, (LPARAM)this);  return lRet;  }  static int CALLBACK FontProc(ENUMLOGFONT \*lpelf,  NEWTEXTMETRIC \*lpntm, int nFontType, LPARAM lParam)  {  CLeftView\* pList = (CLeftView\*)lParam;  // TrueType폰트명만 리스트에 추가  if(nFontType & TRUETYPE\_FONTTYPE)  pList->AddString(lpelf->elfLogFont.lfFaceName);  return 1;  }  }; |  |  | | --- | | // RightView.h  class CRightView : public CWindowImpl<CRightView>  {  CString m\_strFontName;  public:  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CRightView)  MSG\_WM\_PAINT(OnPaint)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  if(!m\_strFontName.IsEmpty()){  // 사이즈20포인트의 폰트를 작성해 , 선택  CFont font;  font.CreatePointFont(200, m\_strFontName);  HFONT hOldFont = dc.SelectFont(font);  // 뷰윈드우에 폰트명을 묘화  CRect rect;  GetClientRect(rect);  dc.SetBkMode(TRANSPARENT);  dc.DrawText(m\_strFontName, m\_strFontName.GetLength(),  rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  }  }  void SetFontName(LPCTSTR lpszFontName){  m\_strFontName = lpszFontName;  Invalidate();  }  }; |  |  | | --- | | // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CSplitterWindow m\_wndSplitter; // 스플리터 윈도우  CLeftView m\_viewLeft; // 왼쪽 페인  CRightView m\_viewRight; // 오른쪽 페인  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  // 왼쪽 페인의 것PreTranslateMessage을 호출하는  if(m\_viewLeft.PreTranslateMessage(pMsg))  return TRUE;  // 오른쪽 페인의 것PreTranslateMessage을 호출하는  return m\_viewRight.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_CODE\_HANDLER\_EX(LBN\_SELCHANGE, OnListSelChange)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_CHANGEVIEW, OnMenuChangeView)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnFileExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 스플리터 윈도우를 작성  m\_wndSplitter.Create (m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS);  // 스플리터 윈도우 확장 스타일을 설정  m\_wndSplitter.SetSplitterExtendedStyle(0);  // 왼쪽 페인의 뷰윈드우를 작성  m\_viewLeft.Create(m\_wndSplitter, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN | WS\_VSCROLL |  LBS\_NOINTEGRALHEIGHT | LBS\_NOTIFY | LBS\_WANTKEYBOARDINPUT | LBS\_SORT,  WS\_EX\_CLIENTEDGE);  m\_wndSplitter.SetSplitterPane(SPLIT\_PANE\_LEFT, m\_viewLeft);  // 오른쪽 페인의 뷰윈드우를 작성  m\_viewRight.Create(m\_wndSplitter, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN,  WS\_EX\_CLIENTEDGE);  m\_wndSplitter.SetSplitterPane(SPLIT\_PANE\_RIGHT, m\_viewRight);  m\_hWndClient = m\_wndSplitter;  UpdateLayout();    // 분할 바의 위치를 설정  m\_wndSplitter.SetSplitterPos(120);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnListSelChange(UINT uNotifyCode, int nID, HWND hWndCtl){  // 현재 선택되고 있는 아이템을 취득  int nIndex = m\_viewLeft.GetCurSel();  if(nIndex != LB\_ERR){  CString strText;  m\_viewLeft.GetText(nIndex, strText);    // 취득한 캐릭터 라인을 오른쪽 페인으로 설정  m\_viewRight.SetFontName(strText);  }  }  void OnMenuChangeView(UINT uNotifyCode, int nID, HWND hWndCtl){  if(m\_wndSplitter.GetSinglePaneMode() == SPLIT\_PANE\_RIGHT)  m\_wndSplitter.SetSinglePaneMode(SPLIT\_PANE\_NONE); // 양페인 표시  else  m\_wndSplitter.SetSinglePaneMode(SPLIT\_PANE\_RIGHT); // 오른쪽 페인만 표시  }  void OnFileExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |  |  | | --- | | // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "LeftView.h"  #include "RightView.h"  #include "MainFrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |   　우선 도구모음 자원의 톱 레벨에 있는 [표시] 아래에 [뷰 변경] 메뉴 아이템을 추가해 그 메뉴 아이템에 ID\_MENUITEM\_CHANGEVIEW 라고 하는 ID를 설정합니다.   　다음에 stdafx.h 에서는 CSplitterWindow 클래스를 사용하기 위해서 atlsplit.h 헤더를 인클루드 합니다.   　다음에 스플리터 윈도우의 왼쪽 페인에 표시하는 뷰 윈도우용 클래스를 정의합니다. 위의 예에서는 프로젝트에 LeftView.h라고 하는 헤더 파일을 추가해 거기에 CWindowImpl 클래스로부터 파생한 CLeftView라고 하는 클래스를 정의하고 있습니다. 이 클래스는 리스트 박스 뷰이며 WM\_CREATE 메세지 핸들러로 사용 가능한 폰트의 폰트 명을 리스트에 추가합니다.   　다음에 스플리터 윈도우의 오른쪽 페인에 표시하는 뷰 윈도우용 클래스를 정의합니다. 위의 예에서는 프로젝트에 RightView.h라고 하는 헤더 파일을 추가해 거기에 CWindowImpl클래스로부터 파생한 CRightView라고 하는 클래스를 정의하고 있습니다. 이 클래스는 외부로부터 폰트명의 캐릭터 라인을 m\_strFontName 이라고 하는 멤버 변수에 대입해 그것을 WM\_PAINT 메세지 핸들러로 묘화 합니다.   　다음에 CMainFrame 클래스내에서 CSplitterWindow 클래스와 좌우의 뷰 윈도우 클래스의 인스턴스를 멤버 변수로서 선언합니다. CMainFrame 클래스의 WM\_CREATE 메세지 핸들러에서는 우선 스플리터 윈도우를 작성한 후 스플리터 윈도우 확장 스타일을 0로 설정해 있습니다. 이것은 스플리터 윈도우의 사이즈가 변경되었을 경우에 왼쪽 페인의 폭(수평 분할 스플리터 윈도우의 경우는 상페인의 높이) 을 고정하는 것을 의미합니다. 스플리터 윈도우 확장 스타일에는 다음의 스타일을 지정할 수 있습니다.   * SPLIT\_PROPORTIONAL   스플리터 윈도우의 사이즈가 변경되었을 경우에 양 페인의 사이즈도 변경합니다.   * SPLIT\_RIGHTALIGNED   스플리터 윈도우의 사이즈가 변경되었을 경우에 오른쪽 페인의 폭을 고정합니다.   * SPLIT\_BOTTOMALIGNED   스플리터 윈도우의 사이즈가 변경되었을 경우에 하 페인의 폭을 고정합니다.   * SPLIT\_NONINTERACTIVE   스플리터 윈도우의 분할 바를 이동 불가로 합니다.  디폴트에서는SPLIT\_PROPORTIONAL이 설정되어 있습니다.   　다음에 스플리터 윈도우를 부모로서 각 페인의 뷰 윈도우를 작성해 그것을 SetSplitterPane()로 스플리터 윈도우의 페인으로 설정합니다. SetSplitterPane()의 제1인수에는 페인을 식별하기 위한 다음과 같은 플래그를 지정합니다.   * SPLIT\_PANE\_LEFT   왼쪽 페인을 식별한다.   * SPLIT\_PANE\_RIGHT   오른쪽 페인을 식별한다.   * SPLIT\_PANE\_TOP   상페인을 식별한다.   * SPLIT\_PANE\_BOTTOM   하 페인을 식별한다.  덧붙여 양쪽 모두의 페인을 한 번으로 설정하는 경우는 SetSplitterPanes()이 사용할 수 있습니다. 이것은 제1인수에 왼쪽(또는 상) 페인의 윈도우 핸들, 제2인수에는 오른쪽(또는 하) 페인의 윈도우 핸들을 설정합니다.   　다음에 스플리터 윈도우의 핸들을 m\_hWndClient에 대입해 프레임 윈도우의 아이 윈도우로 해 SetSplitterPos() 그리고 분할 바의 위치(스플리터 윈도우의 좌단으로부터의 거리)를 설정합니다.   　이와 같이 뷰 윈도우(페인) 의 부모는 스플리터 윈도우이며 스플리터 윈도우의 부모는 프레임 윈도우입니다. 그리고 스플리터 윈도우의 페인은 2개 설정할 수 있습니다. 다음에 나타내는 것은 WTL에 의한 스플리터 윈도우 어플리케이션의 이미지도입니다.  http://home.att.ne.jp/banana/akatsuki/doc/atlwtl/atlwtl20-01/03.png  　다음에 왼쪽 페인의 리스트로 아이템을 선택했을 때의 처리를 추가합니다. 아이템을 선택했을 때에 발생하는 LBN\_SELCHANGE 통지 코드를 가지는 커멘드 메세지는 그 부모인 스플리터 윈도우에 보내집니다만 CSplitterWindow 클래스의 base class인 CSplitterWindowImpl 클래스내의 메세지 맵에는 FORWARD\_NOTIFICATIONS()이 엔트리 되고 있기 때문에 커멘드 메세지는 한층 더 그 부모인 프레임 윈도우에 보내집니다. (페인으로 통지 메시지 (WM\_NOTIFY) 가 발생했을 경우도 프레임 윈도우에 보내집니다. ) 이번 예에서는 CMainFrame 클래스내에 통지 코드가 LBN\_SELCHANGE의 커멘드 메세지 핸들러를 추가해 현재 선택되고 있는 폰트명을 취득해 그것을 오른쪽 페인에 설정해 있습니다.   　다음에 ID가 ID\_MENUITEM\_CHANGEVIEW의 커멘드 메세지 핸들러를 추가합니다. 여기에서는 SetSinglePaneMode()으로 양 페인을 표시하는 모드와 오른쪽 페인 만을 표시하는 모드를 새로 바꿉니다. 덧붙여 지정할 수 있는 모드는 SetSplitterPane()의 제1인수로 지정할 수 있는 4개 외에 SPLIT\_PANE\_NONE도 지정할 수 있습니다. SPLIT\_PANE\_NONE을 지정하면 양쪽 모두의 페인이 표시됩니다. |

▣ 스플릿터 윈도우 > 3페인 스플릿터 윈도우

**< 3 페인 스플리터 윈도우 >**  
  
스플리터 윈도우의 페인에 한층 더 스플리터 윈도우를 설정하는 것으로 3개 이상의 페인을 가지는 스플리터 윈도우를 작성할 수가 있습니다.   
  
　이하에 나타내는 것은 수직 분할 스플리터 윈도우의 오른쪽 페인에 수평 분할 스플리터 윈도우를 설정하는 것으로 3개의 페인을 가지는 스플리터 윈도우를 작성하는 예입니다. 왼쪽 페인에는 폰트 타입의 트리를 표시하고 우상 페인에는 트리로 선택된 타입의 사용 가능한 폰트명을 리스트에 표시하며 우하 페인에는 리스트로 선택된 폰트로 폰트명을 표시합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlframe.h>  #include <atlsplit.h> // 스플리터 윈도우를 사용하기 위해(때문에) |

|  |
| --- |
| // FontType.h  class CFontType  {  int m\_nFontType;  public:  enum {ALL, TRUETYPE, RASTER, VECTOR}; // 폰트 타입 식별자  // constructor  CFontType(int nFontType) : m\_nFontType(nFontType)  {}  // nFontType은 폰트를 열거하는 콜백 함수의 제3인수  bool Equal(int nFontType){  if(m\_nFontType == ALL){  return true;  }else if(nFontType & TRUETYPE\_FONTTYPE){  if(m\_nFontType == TRUETYPE)  return true;  }else if(nFontType & RASTER\_FONTTYPE){  if(m\_nFontType == RASTER)  return true;  }else{  if(m\_nFontType == VECTOR)  return true;  }  return false;  }  }; |

|  |
| --- |
| // TreeView.h  class CTreeView : public CWindowImpl<CTreeView, CTreeViewCtrlEx>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CTreeViewCtrlEx::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  BEGIN\_MSG\_MAP\_EX(CTreeView)  MSG\_WM\_CREATE(OnCreate)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetFont(AtlGetDefaultGuiFont());  CTreeItem tiAll = InsertItem(\_T("모두"), TVI\_ROOT, TVI\_LAST);  tiAll.SetData(CFontType::ALL);  CTreeItem tiFont;  tiFont = tiAll.AddTail(\_T("TrueType"), -1);  tiFont.SetData(CFontType::TRUETYPE);  tiFont = tiAll.AddTail(\_T("라스터"), -1);  tiFont.SetData(CFontType::RASTER);  tiFont = tiAll.AddTail(\_T("벡터"), -1);  tiFont.SetData(CFontType::VECTOR);  return lRet;  }  }; |

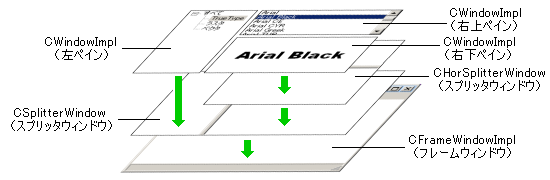
|  |
| --- |
| // ListBoxView.h  class CListBoxView : public CWindowImpl<CListBoxView, CListBox>  {  // EnumFontFamilies()에 건네주는 구조체  struct FONTPROCPARAM{  CListBox list;  CFontType fonttype;  // constructor  FONTPROCPARAM(HWND hWnd, int nFontType)  : list(hWnd), fonttype(nFontType)  {}  };  public:  DECLARE\_WND\_SUPERCLASS(NULL, CListBox::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  BEGIN\_MSG\_MAP\_EX(CListBoxView)  MSG\_WM\_CREATE(OnCreate)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetFont(AtlGetDefaultGuiFont());  return lRet;  }  void FillFontList(int nFontType){  ResetContent();  FONTPROCPARAM fpp(m\_hWnd, nFontType);  CClientDC dc(m\_hWnd);  EnumFontFamilies(dc, NULL,  (FONTENUMPROC)FontProc, (LPARAM)&fpp);  }  static int CALLBACK FontProc(ENUMLOGFONT \*lpelf,  NEWTEXTMETRIC \*lpntm, int nFontType, LPARAM lParam)  {  FONTPROCPARAM\* pFPP = (FONTPROCPARAM\*)lParam;  if(pFPP->fonttype.Equal(nFontType))  pFPP->list.AddString(lpelf->elfLogFont.lfFaceName);  return 1;  }  }; |

|  |
| --- |
| // FontView.h  class CFontView : public CWindowImpl<CFontView>  {  CString m\_strFontName;  public:  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CFontView)  MSG\_WM\_PAINT(OnPaint)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  if(!m\_strFontName.IsEmpty()){  // 사이즈20포인트의 폰트를 작성해 , 선택  CFont font;  font.CreatePointFont(200, m\_strFontName);  HFONT hOldFont = dc.SelectFont(font);  // 뷰윈드우에 폰트명을 묘화  CRect rect;  GetClientRect(rect);  dc.SetBkMode(TRANSPARENT);  dc.DrawText(m\_strFontName, m\_strFontName.GetLength(),  rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  }  }  void SetFontName(LPCTSTR lpszFontName){  m\_strFontName = lpszFontName;  Invalidate();  }  }; |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CSplitterWindow m\_wndVerSplitter; // 수직 분할 스플리터 윈도우  CHorSplitterWindow m\_wndHorSplitter; // 수평 분할 스플리터 윈도우  CTreeView m\_viewTree; // 왼쪽 페인  CListBoxView m\_viewList; // 우상 페인  CFontView m\_viewFont; // 우하 페인  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  // 왼쪽 페인의 것PreTranslateMessage을 호출하는  if(m\_viewTree.PreTranslateMessage(pMsg))  return TRUE;  // 우상 페인의 것PreTranslateMessage을 호출하는  if(m\_viewList.PreTranslateMessage(pMsg))  return TRUE;  // 우하 페인의 것PreTranslateMessage을 호출하는  return m\_viewFont.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  NOTIFY\_CODE\_HANDLER\_EX(NM\_CLICK, OnTreeClick)  COMMAND\_CODE\_HANDLER\_EX(LBN\_SELCHANGE, OnListSelChange)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnFileExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 수직 분할 스플리터 윈도우를 작성  m\_wndVerSplitter.Create (m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS);  m\_wndVerSplitter.SetSplitterExtendedStyle(0);  // 트리뷰윈드우를 작성  m\_viewTree.Create(m\_wndVerSplitter, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN |  TVS\_HASBUTTONS | TVS\_HASLINES | TVS\_LINESATROOT | TVS\_SHOWSELALWAYS,  WS\_EX\_CLIENTEDGE);  // 수직 분할 스플리터 윈도우의 왼쪽 페인에 트리뷰윈드우를 설정  m\_wndVerSplitter.SetSplitterPane(SPLIT\_PANE\_LEFT, m\_viewTree);  // 수평 분할 스플리터 윈도우를 작성  m\_wndHorSplitter.Create (m\_wndVerSplitter, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS);  m\_wndHorSplitter.SetSplitterExtendedStyle(0);  // 리스트복스뷰윈드우를 작성  m\_viewList.Create(m\_wndHorSplitter, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN | WS\_VSCROLL |  LBS\_NOINTEGRALHEIGHT | LBS\_NOTIFY | LBS\_WANTKEYBOARDINPUT | LBS\_SORT,  WS\_EX\_CLIENTEDGE);  // 폰트뷰윈드우를 작성  m\_viewFont.Create(m\_wndHorSplitter, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN,  WS\_EX\_CLIENTEDGE);  // 수평 분할 스플리터 윈도우 상페인에 리스트복스뷰윈드우를 ,  // 하페인에 폰트뷰윈드우를 설정  m\_wndHorSplitter.SetSplitterPanes(m\_viewList, m\_viewFont);  // 수직 분할 스플리터 윈도우의 오른쪽 페인에 수평 분할 스플리터 윈도우를 설정  m\_wndVerSplitter.SetSplitterPane(SPLIT\_PANE\_RIGHT, m\_wndHorSplitter);  m\_hWndClient = m\_wndVerSplitter;  UpdateLayout();    // 분할 바의 위치를 설정  m\_wndVerSplitter.SetSplitterPos(100);  m\_wndHorSplitter.SetSplitterPos(70);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  LRESULT OnTreeClick(LPNMHDR pnmh){  // 클릭된 아이템 핸들 취득  UINT uFlag;  CPoint pt = ::GetMessagePos();  m\_viewTree.ScreenToClient(&pt);  CTreeItem tiItem = m\_viewTree.HitTest(pt, &uFlag);  if(!tiItem.IsNull() && (uFlag & TVHT\_ONITEM))  m\_viewList.FillFontList(tiItem.GetData());  return 0;  }  void OnListSelChange(UINT uNotifyCode, int nID, HWND hWndCtl){  // 현재 선택되고 있는 아이템을 취득해 표시  int nIndex = m\_viewList.GetCurSel();  if(nIndex != LB\_ERR){  CString strText;  m\_viewList.GetText(nIndex, strText);  m\_viewFont.SetFontName(strText);  }  }  void OnFileExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "FontType.h"  #include "TreeView.h"  #include "ListBoxView.h"  #include "FontView.h"  #include "MainFrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

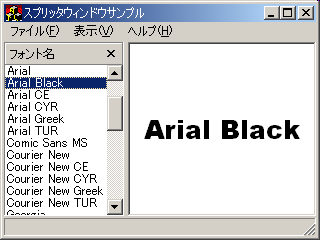
　우선 stdafx.h 에서는 CSplitterWindow, CHorSplitterWindow 클래스를 사용하기 위해서 atlsplit.h 헤더를 인클루드 합니다.   
  
　다음에 폰트 타입의 종류를 캡슐화하기 위한 CFontType 클래스를 정의합니다. 이번 예로 표시하는 폰트 타입은 「모든 것」 「TrueType」 「라스터」 「벡터」의 4종류이므로 enum 그리고 각각의 식별자를 정의하고 있습니다. Equal()라고 하는 멤버 함수는 폰트를 열거하는 콜백 함수로부터 불려 인수로 지정된 폰트 타입과 CFontType 클래스가 보관 유지하고 있는 폰트 타입을 비교합니다.   
  
　다음에 수직 분할 스플리터 윈도우의 왼쪽 페인에 표시하는 뷰 윈도우용 클래스를 정의합니다. 위의 예에서는 프로젝트에 TreeView.h라고 하는 헤더 파일을 추가해 거기에 CWindowImpl 클래스로부터 파생한 CTreeView라고 하는 클래스를 정의하고 있습니다. 이 클래스는 확장 트리뷰를 베이스로 하고 있어 WM\_CREATE 메세지 핸들러로 폰트 타입명의 아이템을 트리에 추가합니다. 이 때 각 아이템에는 SetData()에 의해 폰트 타입 식별자를 설정합니다.   
  
　다음에 수평 분할 스플리터 윈도우 상 페인에 표시하는 뷰 윈도우용 클래스를 정의합니다. 위의 예에서는 프로젝트에 ListBoxView.h라고 하는 헤더 파일을 추가해 거기에 CWindowImpl클래스로부터 파생한 CListBoxView라고 하는 클래스를 정의하고 있습니다. 이 클래스는 리스트 박스를 베이스로 하고 있어 FillFontList()라고 하는 독자적인 멤버 함수로 지정된 폰트 타입의 폰트를 리스트에 추가합니다.   
  
　다음에 수평 분할 스플리터 윈도우 하 페인에 표시하는 뷰 윈도우용 클래스를 정의합니다. 위의 예에서는 프로젝트에 FontView.h라고 하는 헤더 파일을 추가해 거기에 CWindowImpl 클래스로부터 파생한 CFontView라고 하는 클래스를 정의하고 있습니다. 이 클래스는 외부로부터 폰트명의 캐릭터 라인을 m\_strFontName 이라고 하는 멤버 변수에 대입해 그것을 WM\_PAINT 메세지 핸들러로 묘화 합니다.   
  
　다음에 CMainFrame 클래스내에서 수직/수평 분할 스플리터 윈도우 클래스와 각 페인의 뷰 윈도우 클래스의 인스턴스를 멤버 변수로서 선언합니다. CMainFrame 클래스의 WM\_CREATE 메세지 핸들러에서는 우선 수직 분할 스플리터 윈도우를 작성해 그 왼쪽 페인에 뷰 윈도우를 설정합니다. 한층 더 수직 분할 스플리터 윈도우를 부모로서 수평 분할 스플리터 윈도우를 작성해 상하의 페인에 뷰윈드우를 설정합니다. 그리고 이 수평 분할 스플리터 윈도우를 수직 분할 스플리터 윈도우의 오른쪽 페인으로 설정합니다. 다음에 나타내는 것은 3페인 스플리터 윈드우의 이미지도입니다.



　다음에 왼쪽 페인의 트리 아이템을 선택했을 때의 처리를 추가합니다. 이번 예에서는CMainFrame 클래스 내에 통지 코드가 NM\_CLICK 의 통지 메세지 핸들러를 추가해 현재 선택되고 있는 아이템으로 설정되어 있는 데이터(폰트 타입 식별자)를 취득해 그것을FillFontList()에 건네주고 있습니다.   
  
　다음에 우상 페인의 리스트로 아이템을 선택했을 때의 처리를 추가합니다. 이번 예에서는CMainFrame 클래스 내에 통지 코드가 LBN\_SELCHANGE 의 커멘드 메세지 핸들러를 추가해 현재 선택되고 있는 폰트명을 취득해 그것을 우하 페인에 설정해 있습니다.

▣ 스플릿터 윈도우 > 페인 컨테이너

**< 페인 컨테이너 >**  
페인 컨테이너란 Explorer의 왼쪽 페인과 같은 외관을 가지는 뷰 윈도우의 컨테이너입니다. 페인 컨테이너에의 상부에는 타이틀 캐릭터 라인을 설정할 수가 있어 우상에는 페인 컨테이너 윈도우를 닫기 위한 버튼이 붙어 있습니다. WTL은 페인 컨테이너를 작성하기 위해서 CPaneContainer 클래스를 준비해 있습니다.   
  
　이하에 나타내는 것은 CPaneContainer 클래스를 사용해 페인 컨테이너를 작성하는 예입니다. 이전 작성한 스플리터 윈도우의 왼쪽 페인을 페인 컨테이너로 합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atlctrls.h>  #include <atlctrlx.h> // 페인콘테나를 사용하기 위해(때문에)  #include <atlframe.h>  #include <atlsplit.h> |

|  |
| --- |
| // LeftView.h  class CLeftView : public CWindowImpl<CLeftView, CListBox>  {  public:  DECLARE\_WND\_SUPERCLASS(NULL, CListBox::GetWndClassName())  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CLeftView)  MSG\_WM\_CREATE(OnCreate)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  LRESULT lRet = DefWindowProc();  SetFont(AtlGetDefaultGuiFont());  CClientDC dc(m\_hWnd);  EnumFontFamilies(dc, NULL,  (FONTENUMPROC)FontProc, (LPARAM)this);  return lRet;  }  static int CALLBACK FontProc(ENUMLOGFONT \*lpelf,  NEWTEXTMETRIC \*lpntm, int nFontType, LPARAM lParam)  {  CLeftView\* pList = (CLeftView\*)lParam;  // TrueType폰트명만 리스트에 추가  if(nFontType & TRUETYPE\_FONTTYPE)  pList->AddString(lpelf->elfLogFont.lfFaceName);  return 1;  }  }; |

|  |
| --- |
| // RightView.h  class CRightView : public CWindowImpl<CRightView>  {  CString m\_strFontName;  public:  BOOL PreTranslateMessage(MSG\* pMsg){  return FALSE;  }  // 메세지 맵  BEGIN\_MSG\_MAP\_EX(CRightView)  MSG\_WM\_PAINT(OnPaint)  END\_MSG\_MAP()  void OnPaint(HDC /\*hDC\*/){  CPaintDC dc(m\_hWnd);  if(!m\_strFontName.IsEmpty()){  // 사이즈20포인트의 폰트를 작성해 , 선택  CFont font;  font.CreatePointFont(200, m\_strFontName);  HFONT hOldFont = dc.SelectFont(font);  // 뷰윈드우에 폰트명을 묘화  CRect rect;  GetClientRect(rect);  dc.SetBkMode(TRANSPARENT);  dc.DrawText(m\_strFontName, m\_strFontName.GetLength(),  rect, DT\_CENTER | DT\_VCENTER | DT\_SINGLELINE);  // 원의 폰트를 선택  dc.SelectFont(hOldFont);  }  }  void SetFontName(LPCTSTR lpszFontName){  m\_strFontName = lpszFontName;  Invalidate();  }  }; |

|  |
| --- |
| // mainfrm.h  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CSplitterWindow m\_wndSplitter; // 스플리터 윈도우  CPaneContainer m\_painContainer; // 페인콘테나  CLeftView m\_viewLeft; // 왼쪽 페인  CRightView m\_viewRight; // 오른쪽 페인  virtual BOOL PreTranslateMessage(MSG\* pMsg){  if(CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg))  return TRUE;  // 왼쪽 페인의 것PreTranslateMessage을 호출하는  if(m\_viewLeft.PreTranslateMessage(pMsg))  return TRUE;  // 오른쪽 페인의 것PreTranslateMessage을 호출하는  return m\_viewRight.PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateStatusBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_CODE\_HANDLER\_EX(LBN\_SELCHANGE, OnListSelChange)  COMMAND\_ID\_HANDLER\_EX(ID\_PANE\_CLOSE, OnMenuChangeView)  COMMAND\_ID\_HANDLER\_EX(ID\_MENUITEM\_CHANGEVIEW, OnMenuChangeView)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnFileExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 상태 바를 작성  CreateSimpleStatusBar();  UIAddStatusBar(m\_hWndStatusBar);  // 스플리터 윈도우를 작성  m\_wndSplitter.Create (m\_hWnd, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPCHILDREN | WS\_CLIPSIBLINGS);  // 스플리터 윈도우 확장 스타일을 설정  m\_wndSplitter.SetSplitterExtendedStyle(0);  // 페인콘테나를 작성  m\_painContainer.Create(m\_wndSplitter, \_T("폰트 리스트"));  // 왼쪽 페인의 뷰윈드우를 작성  m\_viewLeft.Create(m\_painContainer, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN | WS\_VSCROLL |  LBS\_NOINTEGRALHEIGHT | LBS\_NOTIFY | LBS\_WANTKEYBOARDINPUT | LBS\_SORT,  WS\_EX\_STATICEDGE);  m\_painContainer.SetClient(m\_viewLeft);  m\_wndSplitter.SetSplitterPane(SPLIT\_PANE\_LEFT, m\_painContainer);  // 오른쪽 페인의 뷰윈드우를 작성  m\_viewRight.Create(m\_wndSplitter, rcDefault, NULL,  WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | WS\_CLIPCHILDREN,  WS\_EX\_CLIENTEDGE);  m\_wndSplitter.SetSplitterPane(SPLIT\_PANE\_RIGHT, m\_viewRight);  m\_hWndClient = m\_wndSplitter;  UpdateLayout();    // 분할 바의 위치를 설정  m\_wndSplitter.SetSplitterPos(120);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnListSelChange(UINT uNotifyCode, int nID, HWND hWndCtl){  // 현재 선택되고 있는 아이템을 취득  int nIndex = m\_viewLeft.GetCurSel();  if(nIndex != LB\_ERR){  CString strText;  m\_viewLeft.GetText(nIndex, strText);    // 취득한 캐릭터 라인을 오른쪽 페인으로 설정  m\_viewRight.SetFontName(strText);  }  }  void OnMenuChangeView(UINT uNotifyCode, int nID, HWND hWndCtl){  if(m\_wndSplitter.GetSinglePaneMode() == SPLIT\_PANE\_RIGHT)  m\_wndSplitter.SetSinglePaneMode(SPLIT\_PANE\_NONE); // 양페인 표시  else  m\_wndSplitter.SetSinglePaneMode(SPLIT\_PANE\_RIGHT); // 오른쪽 페인만 표시  }  void OnFileExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "LeftView.h"  #include "RightView.h"  #include "MainFrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 stdafx.h 에서는 CPaneContainer 클래스를 사용하기 위해서 atlctrlx.h 헤더를 인클루드 합니다.   
  
　다음에 CMainFrame 클래스내에서 CPaneContainer 클래스의 인스턴스를 멤버 변수에 추가합니다. CMainFrame 클래스의 WM\_CREATE 메세지 핸들러에서는 우선 스플리터 윈도우를 부모로서 페인 컨테이너를 작성합니다. 이 때 Create()의 제1인수에는 부모 윈도우 핸들, 제2인수에는 페인 컨테이너에 표시하는 타이틀 캐릭터 라인을 지정합니다. 덧붙여 제2인수는 캐릭터 라인 리소스 ID에서도 괜찮습니다. 이 외 Create()에는 윈도우 스타일, 확장 윈도우 스타일, 윈도우 식별자, CREATESTRUCT구조체에 포함되어 있는 윈도우 작성 데이터에의 포인터를 지정할 수 있습니다.   
  
덧붙여 SetPaneContainerExtendedStyle()를 호출하면 페인 컨테이너 확장 스타일을 설정할 수가 있습니다. 다음의 스타일을 지정할 수 있습니다.

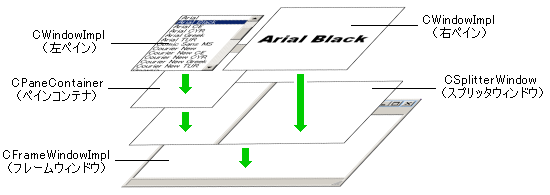
* PANECNT\_NOCLOSEBUTTON

페인 컨테이너의 오른쪽 위의 [닫는] 버튼을 비 표시로 합니다.

* PANECNT\_VERTICAL

페인 컨테이너의 헤더(타이틀이 표시되는 부분)를 수직에 표시합니다.

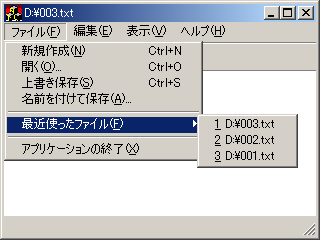
　다음에 페인 컨테이너를 부모로서 왼쪽 페인의 뷰 윈도우를 작성해 그 뷰 윈도우의 핸들을 SetClient을 호출해 페인 컨테이너에 세트 합니다. 한층 더 그 페인 컨테이너를 SetSplitterPane()에 의해 스플리터 윈도우의 왼쪽 페인에 세트 합니다.   
  
　이와 같이 뷰 윈도우의 부모는 페인 컨테이너이며 페인 컨테이너의 부모는 스플리터 윈도우이며 스플리터 윈도우의 부모는 프레임 윈도우입니다. 다음에 나타내는 것은 WTL의 스플리터 윈도우와 페인 컨테이너를 사용한 어플리케이션의 이미지도입니다.



　마지막으로 페인 컨테이너의 오른쪽 위에 있는 [닫는]버튼을 눌렀을 때의 처리를 추가합니다. 이 버튼을 누르면 ID가 ID\_PANE\_CLOSE의 커멘드 메세지가 발생하기 때문에 CMainFrame 클래스의 메세지 맵에 엔트리를 추가해 OnMenuChangeView()를 호출하고 있습니다. 이 함수는 이전 작성한 것으로 뷰의 표시를 새로 바꿉니다.   
  
　그런데 페인 컨테이너의 메세지 맵에도 스플리터 윈도우의 경우와 같게FORWARD\_NOTIFICATIONS()가 엔트리 되고 있습니다. 이 때문에 뷰 윈도우로 발생한 커멘드 메세지나 통지 메세지는 그 부모인 페인 컨테이너에 보내져 한층 더 스플리터 윈도우에 전송 되어 마지막에 프레임 윈도우에 닿습니다.

▣ MRU 파일 리스트 > MRU 파일 리스트

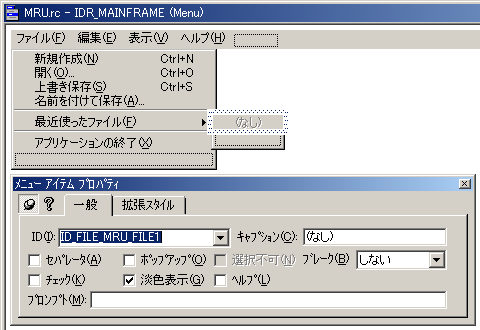
**< MRU 파일 리스트 >**  
  
MRU 파일 리스트는 [최근 사용한 파일]리스트입니다. 통상 이 리스트는 메뉴에 추가되어 어플리케이션이 새로운 파일을 열 때마다 갱신됩니다. WTL은 MRU 파일 리스트를 작성하기 위해서 CRecentDocumentList 클래스를 준비해 있습니다.   
  
　이하에 나타내는 것은 CRecentDocumentList 클래스를 사용해 MRU 파일 리스트를 작성하는 예입니다. 프레임 윈도우의 메뉴에 [최근 사용한 파일]을 추가해 [여는]을 실행해 새로운 파일을 선택할 때마다 MRU 파일 리스트를 갱신합니다.



|  |
| --- |
| // stdafx.h  #include <atlbase.h>  #include <atlapp.h>  extern CAppModule \_Module;  #include <atlwin.h>  #include <atlcrack.h>  #include <atlmisc.h>  #include <atldlgs.h>  #include <atlframe.h> |

|  |
| --- |
| // mainfrm.h  LPCTSTR g\_lpszMRURegKey = \_T("Software\\WTL Samples");  class CMainFrame : public CFrameWindowImpl<CMainFrame>,  public CUpdateUI<CMainFrame>, public CMessageFilter, public CIdleHandler  {  public:  DECLARE\_FRAME\_WND\_CLASS(NULL, IDR\_MAINFRAME)  CRecentDocumentList m\_mru;  virtual BOOL PreTranslateMessage(MSG\* pMsg){  return CFrameWindowImpl<CMainFrame>::PreTranslateMessage(pMsg);  }  virtual BOOL OnIdle(){  UIUpdateToolBar();  return FALSE;  }  BEGIN\_UPDATE\_UI\_MAP(CMainFrame)  // 엔트리 없음  END\_UPDATE\_UI\_MAP()  BEGIN\_MSG\_MAP\_EX(CMainFrame)  MSG\_WM\_CREATE(OnCreate)  COMMAND\_ID\_HANDLER\_EX(ID\_FILE\_OPEN, OnFileOpen)  COMMAND\_RANGE\_HANDLER\_EX(ID\_FILE\_MRU\_FIRST, ID\_FILE\_MRU\_LAST, OnFileRecent)  COMMAND\_ID\_HANDLER\_EX(ID\_APP\_EXIT, OnAppExit)  CHAIN\_MSG\_MAP(CUpdateUI<CMainFrame>)  CHAIN\_MSG\_MAP(CFrameWindowImpl<CMainFrame>)  END\_MSG\_MAP()  LRESULT OnCreate(LPCREATESTRUCT lpcs){  // 리버를 작성  CreateSimpleReBar();  // 툴바를 작성해 밴드에 추가  HWND hWndToolBar = CreateSimpleToolBarCtrl(m\_hWnd,  IDR\_MAINFRAME, FALSE, ATL\_SIMPLE\_TOOLBAR\_PANE\_STYLE);  AddSimpleReBarBand(hWndToolBar);  UIAddToolBar(hWndToolBar);  // 상태 바를 작성  CreateSimpleStatusBar();  // MRU파일 리스트를 설정  CMenuHandle menu = GetMenu();  CMenuHandle menuFile = menu.GetSubMenu(0);  CMenuHandle menuMru = menuFile.GetSubMenu(5);  m\_mru.SetMenuHandle(menuMru);  m\_mru.SetMaxEntries(5);  m\_mru.ReadFromRegistry(g\_lpszMRURegKey);  // 메시지 루프에 메세지 필터와 아이돌 핸들러를 추가  CMessageLoop\* pLoop = \_Module.GetMessageLoop();  pLoop->AddMessageFilter(this);  pLoop->AddIdleHandler(this);  return 0;  }  void OnFileOpen(UINT uNotifyCode, int nID, HWND hWndCtl){  CFileDialog dlg(TRUE, NULL, NULL, OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT,  \_T("모든 파일 (\*.\*)\0\*.\*\0"), m\_hWnd);  if(dlg.DoModal() == IDOK){  // MRU파일 리스트에 아이템(파일 패스) 을 추가  m\_mru.AddToList(dlg.m\_ofn.lpstrFile);  SetWindowText(dlg.m\_ofn.lpstrFile);  // MRU파일 리스트를 레지스트리에 기입하는  m\_mru.WriteToRegistry(g\_lpszMRURegKey);  }  }  void OnFileRecent(UINT uNotifyCode, int nID, HWND hWndCtl){  CString strFile;  if(m\_mru.GetFromList(nID, strFile)){  CFindFile find;  if(find.FindFile(strFile)){  // ID으로 지정한 아이템을MRU파일 리스트의 선두에 이동  m\_mru.MoveToTop(nID);  SetWindowText(strFile);  }else{  // ID으로 지정한 아이템을MRU파일 리스트로부터 삭제  m\_mru.RemoveFromList(nID);  }  // MRU파일 리스트를 레지스트리에 기입하는  m\_mru.WriteToRegistry(g\_lpszMRURegKey);  }  }  void OnAppExit(UINT uNotifyCode, int nID, HWND hWndCtl){  PostMessage(WM\_CLOSE);  }  }; |

|  |
| --- |
| // app.cpp  #include "stdafx.h"  #include "resource.h"  #include "MainFrm.h"  CAppModule \_Module;  int WINAPI \_tWinMain(HINSTANCE hInstance, HINSTANCE, LPTSTR lpCmdLine, int nCmdShow)  {  HRESULT hRes = ::CoInitialize(NULL);  ATLASSERT(SUCCEEDED(hRes));  ::DefWindowProc(NULL, 0, 0, 0L);  AtlInitCommonControls(ICC\_COOL\_CLASSES | ICC\_WIN95\_CLASSES);  hRes = \_Module.Init(NULL, hInstance);  ATLASSERT(SUCCEEDED(hRes));  int nRet = 0;  // BLOCK: 어플리케이션 실행  {  CMessageLoop theLoop;  \_Module.AddMessageLoop(&theLoop);  CMainFrame wnd;  wnd.CreateEx();  wnd.ShowWindow(nCmdShow);  wnd.UpdateWindow();  nRet = theLoop.Run();  \_Module.RemoveMessageLoop();  }  \_Module.Term();  ::CoUninitialize();  return nRet;  } |

　우선 도구모음 자원의 [파일] 이래에 [최근 사용한 파일]을 추가해 그 아래에 [(없음)] 라고 하는 메뉴 아이템을 추가하는 이 메뉴 아이템에는ID\_FILE\_MRU\_FILE1이라고 하는 ID를 설정해 [엷은 색 표시]에 체크를 넣습니다.   
  
  
  
ID\_FILE\_MRU\_FILE1라고 하는 ID는 WTL의 atlres.h 헤더에 정의되고 있습니다. WTL은 MRU 파일 리스트를 위한 ID를 16개(ID\_FILE\_MRU\_FILE16)까지 준비해 있습니다. CRecentDocumentList 클래스는 이러한 ID를 사용해 MRU 파일 리스트내의 각 아이템을 관리합니다.   
  
　다음에 stdafx.h 에서는 CRecentDocumentList 클래스를 사용하기 위해서 atlmisc.h 헤더를 인클루드 합니다.   
  
　다음에 mainfrm.h 에서 글로벌 변수로서 "Software\\WTL Samples" 라고 하는 캐릭터 라인을 정의합니다. 이것은 레지스트리의 HKEY\_CURRENT\_USER에 계속되는 키를 나타냅니다. CRecentDocumentList 클래스는 MRU 파일 리스트를 레지스트리에 보존합니다.   
  
　CMainFrame 클래스내에서는 CRecentDocumentList 클래스의 인스턴스를 멤버 변수에 추가합니다. CMainFrame 클래스의 WM\_CREATE 메세지 핸들러에서는 각 바를 작성한 후 에 MRU 파일 리스트를 설정합니다. 우선 CRecentDocumentList 클래스의 멤버 함수인 SetMenuHandle()을 호출해 MRU 파일 리스트를 삽입하는 위치의 메뉴 핸들을 설정합니다. 한층 더 SetMaxEntries()를 호출해 리스트에 추가하는 아이템수를 설정해 ReadFromRegistry()를 호출해 레지스트리로부터 MRU 파일 리스트를 읽어들입니다. 덧붙여 SetMaxEntries()로 아이템수를 설정하지 않으면 초기치로서 4가 설정됩니다.   
  
　다음에 [여는]을 실행했을 때에 불려지는 커멘드 메세지 핸들러 OnFileOpen를 정의합니다. 여기에서는 [파일을 여는] 다이얼로그를 열어 선택된 파일의 패스를 AddToList()을 호출해 MRU 파일 리스트에 추가합니다. 그 후 파일 패스를 타이틀 바로 설정하고 나서 WriteToRegistry()를 호출해 MRU 파일 리스트를 레지스트리에 기입합니다.   
  
메뉴의 [최근 사용한 파일]로부터 아이템을 선택했을 때에 불려지는 커멘드 메세지 핸들러 OnFileRecent()를 정의합니다. 이 핸들러는 메세지 맵의 COMMAND\_RANGE\_HANDLER\_EX 매크로에 의해 매핑 되어 ID의 범위가 ID\_FILE\_MRU\_FIRST로부터 ID\_FILE\_MRU\_LAST까지의 커멘드 메세지를 처리합니다. 이러한 ID는 atlres.h 헤더로 정의되고 있어 각각 ID\_FILE\_MRU\_FILE1, ID\_FILE\_MRU\_FILE16이라고 같은 값입니다. OnFileRecent() 에서는 GetFromList()을 호출하는 것에 의해 MRU 파일 리스트로부터 아이템(파일 패스)을 취득해 그 파일이 존재했을 경우는 MoveToTop()을 호출해 아이템을 MRU 파일 리스트의 선두로 이동하고 나서 타이틀 바에 파일 패스를 설정합니다. 파일이 존재하지 않았던 경우는 RemoveFromList()을 호출해 MRU 리스트로부터 아이템을 삭제합니다.   
  
　이와 같이 CRecentDocumentList 클래스는 ID\_FILE\_MRU\_FIRST(ID\_FILE\_MRU\_FILE1)으로부터 ID\_FILE\_MRU\_LAST(ID\_FILE\_MRU\_FILE16)까지의 16개의 아이템을 관리할 수 있습니다. 덧붙여 CRecentDocumentList 클래스에서 이러한 ID가 사용되는 것은 CRecentDocumentList 클래스 및 base class가 다음과 같이 정의되고 있기 때문입니다. 템플릿의 디폴트 인수로서 ID\_FILE\_MRU\_FIRST와 ID\_FILE\_MRU\_LAST가 지정되어 있는 것에 주목해 주세요.

|  |
| --- |
| // atlmisc.h  template <class T, int t\_cchItemLen = MAX\_PATH,  int t\_nFirstID = ID\_FILE\_MRU\_FIRST, int t\_nLastID = ID\_FILE\_MRU\_LAST>  class CRecentDocumentListBase  {  ...  };  class CRecentDocumentList : public CRecentDocumentListBase<CRecentDocumentList>  {  public:  // nothing here  }; |

이 때문에 MRU 파일 리스트에 16개 이상의 아이템을 추가하고 싶은 경우는 CRecentDocumentListBase 클래스의 상속 클래스를 작성해 템플릿 인수에 스스로 정의한 연속하는 ID을 지정할 필요가 있습니다. 다음에 나타내는 것은 최대32개의 아이템을 추가할 수 있는 독자적인 MRU 파일 리스트 클래스를 정의하는 예입니다.

|  |
| --- |
| // ID\_MY\_1 과ID\_MY\_32를 스스로 정의  // ID\_MY\_32는 (ID\_MY\_1 + 32)  class CMyRDL : public CRecentDocumentListBase<CMyRDL, MAX\_PATH, ID\_MY\_1, ID\_MY\_32>  {  }; |

▣ 부록 > WTL 메시지 매크로 일람

|  |
| --- |
| **WTL메세지 매크로 일람** |
| 이하에 나타내는 것은 WTL의 atlcrack.h에 준비되어 있는 메세지 매크로와 거기에 대응하는 핸들러 함수 prototype의 일람입니다. 덧붙여 메세지 핸들러의 함수명이나 인수명은 MFC를 참고로 하고 있습니다.   |  |  | | --- | --- | | 메세지 매크로명 | 메세지 핸들러 함수 prototype | | MSG\_WM\_CREATE | LRESULT OnCreate(LPCREATESTRUCT lpCreateStruct); | | MSG\_WM\_INITDIALOG | LRESULT OnInitDialog(HWND hWnd, LPARAM lParam); | | MSG\_WM\_COPYDATA | LRESULT OnCopyData(HWND hWnd, PCOPYDATASTRUCT pCopyDataStruct); | | MSG\_WM\_DESTROY | void OnDestroy(); | | MSG\_WM\_MOVE | void OnMove(CPoint point); | | MSG\_WM\_SIZE | void OnSize(UINT nType, CSize size); | | MSG\_WM\_ACTIVATE | void OnActivate(UINT nState, BOOL bMinimized, HWND hWnd); | | MSG\_WM\_SETFOCUS | void OnSetFocus(HWND hWnd); | | MSG\_WM\_KILLFOCUS | void OnKillFocus(HWND hWnd); | | MSG\_WM\_ENABLE | void OnEnable(BOOL bEnable); | | MSG\_WM\_PAINT | void OnPaint(HDC /\*hDC\*/); | | MSG\_WM\_CLOSE | void OnClose(); | | MSG\_WM\_QUERYENDSESSION | LRESULT OnQueryEndSession(UINT reserved, UINT nOption); | | MSG\_WM\_QUERYOPEN | LRESULT OnQueryOpen(); | | MSG\_WM\_ERASEBKGND | LRESULT OnEraseBkgnd(HDC hDC); | | MSG\_WM\_SYSCOLORCHANGE | void OnSysColorChange(); | | MSG\_WM\_ENDSESSION | void OnEndSession(BOOL bEnding, UINT nOption); | | MSG\_WM\_SHOWWINDOW | void OnShowWindow(BOOL bShow, int nStatus); | | MSG\_WM\_CTLCOLOREDIT | LRESULT OnCtlColorEdit(HDC hDC, HWND hWnd); | | MSG\_WM\_CTLCOLORLISTBOX | LRESULT OnCtlColorListbox(HDC hDC, HWND hWnd); | | MSG\_WM\_CTLCOLORBTN | LRESULT OnCtlColorBtn(HDC hDC, HWND hWnd); | | MSG\_WM\_CTLCOLORDLG | LRESULT OnCtlColorDlg(HDC hDC, HWND hWnd); | | MSG\_WM\_CTLCOLORSCROLLBAR | LRESULT OnCtlColorScrollbar(HDC hDC, HWND hWnd); | | MSG\_WM\_CTLCOLORSTATIC | LRESULT OnCtlColorStatic(HDC hDC, HWND hWnd); | | MSG\_WM\_SETTINGCHANGE | void OnSettingChange(UINT nFlag, LPCTSTR lpSetting); | | MSG\_WM\_DEVMODECHANGE | void OnDevModeChange(LPCTSTR lpDeviceName); | | MSG\_WM\_ACTIVATEAPP | void OnActivateApp(BOOL bActive, DWORD dwThreadID); | | MSG\_WM\_FONTCHANGE | void OnFontChange(); | | MSG\_WM\_TIMECHANGE | void OnTimeChange(); | | MSG\_WM\_CANCELMODE | void OnCancelMode(); | | MSG\_WM\_SETCURSOR | LRESULT OnSetCursor(HWND hWnd, UINT nHitTest, UINT nMessage); | | MSG\_WM\_MOUSEACTIVATE | LRESULT OnMouseActivate(HWND hWnd, UINT nHitTest, UINT nMessage); | | MSG\_WM\_CHILDACTIVATE | void OnChildActivate(); | | MSG\_WM\_GETMINMAXINFO | void OnGetMinMaxInfo(LPMINMAXINFO lpMMI); | | MSG\_WM\_ICONERASEBKGND | void OnIconEraseBkgnd(HDC hDC); | | MSG\_WM\_SPOOLERSTATUS | void OnSpoolerStatus(UINT nStatus, UINT nJobs); | | MSG\_WM\_DRAWITEM | void OnDrawItem(UINT nIDCtl, LPDRAWITEMSTRUCT lpDrawItemStruct); | | MSG\_WM\_MEASUREITEM | void OnMeasureItem(UINT nIDCtl, LPMEASUREITEMSTRUCT lpMeasureItemStruct); | | MSG\_WM\_DELETEITEM | void OnDeleteItem(UINT nIDCtl, LPDELETEITEMSTRUCT lpDeleteItemStruct); | | MSG\_WM\_CHARTOITEM | LRESULT OnCharToItem(UINT nChar, UINT nIndex, HWND hWnd); | | MSG\_WM\_VKEYTOITEM | LRESULT OnVKeyToItem(UINT nChar, UINT nIndex, HWND hWnd); | | MSG\_WM\_QUERYDRAGICON | LRESULT OnQueryDragIcon(); | | MSG\_WM\_COMPAREITEM | LRESULT OnCompareItem(UINT nIDCtl, LPCOMPAREITEMSTRUCT lpCompareItemStruct); | | MSG\_WM\_COMPACTING | void OnCompacting(UINT nCpuTime); | | MSG\_WM\_NCCREATE | LRESULT OnNcCreate(LPCREATESTRUCT lpCreateStruct); | | MSG\_WM\_NCDESTROY | void OnNcDestroy(); | | MSG\_WM\_NCCALCSIZE | LRESULT OnNcCalcSize(BOOL bCalcValidRects, LPARAM lParam); | | MSG\_WM\_NCHITTEST | LRESULT OnNcHitTest(CPoint point); | | MSG\_WM\_NCPAINT | void OnNcPaint(HRGN hRgn); | | MSG\_WM\_NCACTIVATE | LRESULT OnNcActivate(BOOL bActive); | | MSG\_WM\_GETDLGCODE | LRESULT OnGetDlgCode(LPMSG lpMsg); | | MSG\_WM\_NCMOUSEMOVE | void OnNcMouseMove(UINT nHitTest, CPoint point); | | MSG\_WM\_NCLBUTTONDOWN | void OnNcLButtonDown(UINT nHitTest, CPoint point); | | MSG\_WM\_NCLBUTTONUP | void OnNcLButtonUp(UINT nHitTest, CPoint point); | | MSG\_WM\_NCLBUTTONDBLCLK | void OnNcLButtonDblClk(UINT nHitTest, CPoint point); | | MSG\_WM\_NCRBUTTONDOWN | void OnNcRButtonDown(UINT nHitTest, CPoint point); | | MSG\_WM\_NCRBUTTONUP | void OnNcRButtonUp(UINT nHitTest, CPoint point); | | MSG\_WM\_NCRBUTTONDBLCLK | void OnNcRButtonDblClk(UINT nHitTest, CPoint point); | | MSG\_WM\_NCMBUTTONDOWN | void OnNcMButtonDown(UINT nHitTest, CPoint point); | | MSG\_WM\_NCMBUTTONUP | void OnNcMButtonUp(UINT nHitTest, CPoint point); | | MSG\_WM\_NCMBUTTONDBLCLK | void OnNcMButtonDblClk(UINT nHitTest, CPoint point); | | MSG\_WM\_KEYDOWN | void OnKeyDown(TCHAR vkey, UINT repeats, UINT code); | | MSG\_WM\_KEYUP | void OnKeyUp(TCHAR vkey, UINT repeats, UINT code); | | MSG\_WM\_CHAR | void OnChar(TCHAR vkey, UINT repeats, UINT code); | | MSG\_WM\_DEADCHAR | void OnDeadChar(TCHAR vkey, UINT repeats, UINT code); | | MSG\_WM\_SYSKEYDOWN | void OnSysKeyDown(TCHAR vkey, UINT repeats, UINT code); | | MSG\_WM\_SYSKEYUP | void OnSysKeyUp(TCHAR vkey, UINT repeats, UINT code); | | MSG\_WM\_SYSCHAR | void OnSysChar(TCHAR vkey, UINT repeats, UINT code); | | MSG\_WM\_SYSDEADCHAR | void OnSysDeadChar(TCHAR vkey, UINT repeats, UINT code); | | MSG\_WM\_SYSCOMMAND | void OnSysCommand(UINT nID, CPoint point); | | MSG\_WM\_TCARD | void OnTCard(UINT idAction, DWORD dwActionData); | | MSG\_WM\_TIMER | void OnTimer(UINT nIDEvent, TIMERPROC lpTimerFunc); | | MSG\_WM\_HSCROLL | void OnHScroll(int nSBCode, short nPos, HWND hWnd); | | MSG\_WM\_VSCROLL | void OnVScroll(int nSBCode, short nPos, HWND hWnd); | | MSG\_WM\_INITMENU | void OnInitMenu(HMENU hMenu); | | MSG\_WM\_INITMENUPOPUP | void OnInitMenuPopup(HMENU hMenu, UINT nIndex, BOOL bSysMenu); | | MSG\_WM\_MENUSELECT | void OnMenuSelect(UINT nItemID, UINT nFlags, HMENU hSysMenu); | | MSG\_WM\_MENUCHAR | LRESULT OnMenuChar(TCHAR Char, UINT nFlags, HMENU hMenu); | | MSG\_WM\_NOTIFY | LRESULT OnNotify(int nID, LPNMHDR pnmh); | | MSG\_WM\_ENTERIDLE | void OnEnterIdle(UINT nWhy, HWND hWnd); | | MSG\_WM\_MOUSEMOVE | void OnMouseMove(UINT nFlags, CPoint point); | | MSG\_WM\_MOUSEWHEEL | LRESULT OnMouseWheel(UINT nFlags, short zDelta, CPoint point); | | MSG\_WM\_LBUTTONDOWN | void OnLButtonDown(UINT nFlags, CPoint point); | | MSG\_WM\_LBUTTONUP | void OnLButtonUp(UINT nFlags, CPoint point); | | MSG\_WM\_LBUTTONDBLCLK | void OnLButtonDblClk(UINT nFlags, CPoint point); | | MSG\_WM\_RBUTTONDOWN | void OnRButtonDown(UINT nFlags, CPoint point); | | MSG\_WM\_RBUTTONUP | void OnRButtonUp(UINT nFlags, CPoint point); | | MSG\_WM\_RBUTTONDBLCLK | void OnRButtonDblClk(UINT nFlags, CPoint point); | | MSG\_WM\_MBUTTONDOWN | void OnMButtonDown(UINT nFlags, CPoint point); | | MSG\_WM\_MBUTTONUP | void OnMButtonUp(UINT nFlags, CPoint point); | | MSG\_WM\_MBUTTONDBLCLK | void OnMButtonDblClk(UINT nFlags, CPoint point); | | MSG\_WM\_PARENTNOTIFY | void OnParentNotify(UINT nMsg, UINT nID, LPARAM lParam); | | MSG\_WM\_MDIACTIVATE | void OnMDIActivate(HWND hWndDeactivate, HWND hWndActivate); | | MSG\_WM\_RENDERFORMAT | void OnRenderFormat(UINT nFormat); | | MSG\_WM\_RENDERALLFORMATS | void OnRenderAllFormats(); | | MSG\_WM\_DESTROYCLIPBOARD | void OnDestroyClipboard(); | | MSG\_WM\_DRAWCLIPBOARD | void OnDrawClipboard(); | | MSG\_WM\_PAINTCLIPBOARD | void OnPaintClipboard(HWND hWnd, const LPPAINTSTRUCT pPaintStruct); | | MSG\_WM\_VSCROLLCLIPBOARD | void OnVScrollClipboard(HWND hWnd, UINT nSBCode, UINT nPos); | | MSG\_WM\_CONTEXTMENU | void OnContextMenu(HWND hWnd, CPoint pos); | | MSG\_WM\_SIZECLIPBOARD | void OnSizeClipboard(HWND hWnd, const LPRECT rect); | | MSG\_WM\_ASKCBFORMATNAME | void OnAskCbFormatName(DWORD dwMaxCount, LPTSTR lpszString); | | MSG\_WM\_CHANGECBCHAIN | void OnChangeCbChain(HWND hWndRemove, HWND hWndAfter); | | MSG\_WM\_HSCROLLCLIPBOARD | void OnHScrollClipboard(HWND hWnd, UINT nSBCode, UINT nPos); | | MSG\_WM\_QUERYNEWPALETTE | LRESULT OnQueryNewPalette(); | | MSG\_WM\_PALETTECHANGED | void OnPaletteChanged(HWND hWnd); | | MSG\_WM\_PALETTEISCHANGING | void OnPaletteIsChanging(HWND hWnd); | | MSG\_WM\_DROPFILES | void OnDropFiles(HDROP hDropInfo); | | MSG\_WM\_WINDOWPOSCHANGING | void OnWindowPosChanging(LPWINDOWPOS lpwndpos); | | MSG\_WM\_WINDOWPOSCHANGED | void OnWindowPosChanged(LPWINDOWPOS lpwndpos); | | MSG\_WM\_EXITMENULOOP | void OnExitMenuLoop(BOOL bIsTrackPopupMenu); | | MSG\_WM\_ENTERMENULOOP | void OnEnterMenuLoop(BOOL bIsTrackPopupMenu); | | MSG\_WM\_STYLECHANGED | void OnStyleChanged(UINT nStyleType, LPSTYLESTRUCT lpStyleStruct); | | MSG\_WM\_STYLECHANGING | void OnStyleChanging(UINT nStyleType, LPSTYLESTRUCT lpStyleStruct); | | MSG\_WM\_SIZING | void OnSizing(UINT nSide, LPRECT lpRect); | | MSG\_WM\_MOVING | void OnMoving(UINT nSide, LPRECT lpRect); | | MSG\_WM\_CAPTURECHANGED | void OnCaptureChanged(HWND hWnd); | | MSG\_WM\_DEVICECHANGE | LRESULT OnDeviceChange(UINT nEventType, DWORD dwData); | | MSG\_WM\_COMMAND | void OnCommand(UINT nCode, int nID, HWND hWnd); | | MSG\_WM\_DISPLAYCHANGE | void OnDisplayChange(UINT nDepth, CSize size); | | MSG\_WM\_ENTERSIZEMOVE | void OnEnterSizeMove(); | | MSG\_WM\_EXITSIZEMOVE | void OnExitSizeMove(); | | MSG\_WM\_GETFONT | LRESULT OnGetFont(); | | MSG\_WM\_GETHOTKEY | LRESULT OnGetHotKey(); | | MSG\_WM\_GETICON | LRESULT OnGetIcon(UINT nType); | | MSG\_WM\_GETTEXT | LRESULT OnGetText(int nSize, LPTSTR lpBuffer); | | MSG\_WM\_GETTEXTLENGTH | LRESULT OnGetTextLength(); | | MSG\_WM\_HELP | void OnHelp(LPHELPINFO lpHelpInfo); | | MSG\_WM\_HOTKEY | void OnHotKey(int nID, UINT nOption, UINT nCode); | | MSG\_WM\_INPUTLANGCHANGE | void OnInputLangChange(DWORD dwCharset, HKL hkl); | | MSG\_WM\_INPUTLANGCHANGEREQUEST | void OnInputLangChangeRequest(BOOL bFlag, HKL hkl); | | MSG\_WM\_NEXTDLGCTL | void OnNextDlgCtl(BOOL bUsage, WPARAM wParam); | | MSG\_WM\_NEXTMENU | void OnNextMenu(int nCode, LPMDINEXTMENU lpMDINextMenu); | | MSG\_WM\_NOTIFYFORMAT | LRESULT OnNotifyFormat(HWND hWnd, int nCommand); | | MSG\_WM\_POWERBROADCAST | LRESULT OnPowerBroadcast(DWORD dwEvent, DWORD dwData); | | MSG\_WM\_PRINT | void OnPrint(HDC hDC, UINT nOption); | | MSG\_WM\_PRINTCLIENT | void OnPrintClient(HDC hDC, UINT nOption); | | MSG\_WM\_RASDIALEVENT | void OnRASDialEvent(RASCONNSTATE rasconnstate, DWORD dwError); | | MSG\_WM\_SETFONT | void OnSetFont(HFONT hFont, BOOL bRedraw); | | MSG\_WM\_SETHOTKEY | LRESULT OnSetHotKey(int nCode, UINT nOption); | | MSG\_WM\_SETICON | LRESULT OnSetIcon(UINT nType, HICON hIcon); | | MSG\_WM\_SETREDRAW | void OnSetRedraw(BOOL bRedraw); | | MSG\_WM\_SETTEXT | LRESULT OnSetText(LPCTSTR lpString); | | MSG\_WM\_USERCHANGED | void OnUserChanged(); | | MSG\_WM\_MOUSEHOVER | void OnMouseHover(WPARAM wParam, CPoint point); | | MSG\_WM\_MOUSELEAVE | void OnMouseLeave(); | | MSG\_WM\_MENURBUTTONUP | void OnMenuRButtonUp(WPARAM wParam, HMENU hMenu); | | MSG\_WM\_MENUDRAG | LRESULT OnMenuDrag(WPARAM wParam, HMENU hMenu); | | MSG\_WM\_MENUGETOBJECT | LRESULT OnMenuGetObject(PMENUGETOBJECTINFO pMGOI); | | MSG\_WM\_UNINITMENUPOPUP | void OnUninitMenuPopup(UINT nID, HMENU hMenu); | | MSG\_WM\_MENUCOMMAND | void OnMenuCommand(WPARAM wParam, HMENU hMenu); | | MSG\_WM\_APPCOMMAND | void OnAppCommand(HWND hWnd, short cmd, WORD device, int key); | | MSG\_WM\_NCXBUTTONDOWN | void OnNcXButtonDown(int button, short hittest, CPoint pos); | | MSG\_WM\_NCXBUTTONUP | void OnNcXButtonUp(int button, short hittest, CPoint pos); | | MSG\_WM\_NCXBUTTONDBLCLK | void OnNcXButtonDblClk(int button, short hittest, CPoint pos); | | MSG\_WM\_XBUTTONDOWN | void OnXButtonDown(int button, int key, CPoint pos); | | MSG\_WM\_XBUTTONUP | void OnXButtonUp(int button, int key, CPoint pos); | | MSG\_WM\_XBUTTONDBLCLK | void OnXButtonDblClk(int button, int key, CPoint pos); | | MSG\_WM\_CHANGEUISTATE | void OnChangeUIState(WORD action, WORD flag); | | MSG\_WM\_UPDATEUISTATE | void OnUpdateUIState(WORD action, WORD flag); | | MSG\_WM\_QUERYUISTATE | LRESULT OnQueryUIState(); | | MSG\_WM\_INPUT | void OnInput(WPARAM code, HRAWINPUT hRawInput); | | MSG\_WM\_UNICHAR | void OnUniChar(TCHAR vkey, UINT repeats, UINT code); | | MSG\_WM\_WTSSESSION\_CHANGE | void OnWTSSession\_Change(WPARAM wCode, PWTSSESSION\_NOTIFICATION pWTSSN); | | MSG\_WM\_THEMECHANGED | void OnThemeChanged(); | | MSG\_WM\_FORWARDMSG | LRESULT OnForwardMsg(LPMSG lpMsg, DWORD dwUserData); | | MSG\_DM\_GETDEFID | LRESULT OnGetDefID(); | | MSG\_DM\_SETDEFID | void OnSetDefID(UINT nID); | | MSG\_DM\_REPOSITION | void OnReposition(); | | MSG\_OCM\_COMMAND | void OnCommand(UINT nCode, int nID, HWND hWnd); | | MSG\_OCM\_NOTIFY | LRESULT OnNotify(int nID, LPNMHDR pnmh); | | MSG\_OCM\_PARENTNOTIFY | void OnParentNotify(UINT nMsg, UINT nID, LPARAM lParam); | | MSG\_OCM\_DRAWITEM | void OnDrawItem(UINT nIDCtl, LPDRAWITEMSTRUCT lpDrawItemStruct); | | MSG\_OCM\_MEASUREITEM | void OnMeasureItem(UINT nIDCtl, LPMEASUREITEMSTRUCT lpMeasureItemStruct); | | MSG\_OCM\_COMPAREITEM | LRESULT OnCompareItem(UINT nIDCtl, LPCOMPAREITEMSTRUCT lpCompareItemStruct); | | MSG\_OCM\_DELETEITEM | void OnDeleteItem(UINT nIDCtl, LPDELETEITEMSTRUCT lpDeleteItemStruct); | | MSG\_OCM\_VKEYTOITEM | LRESULT OnVKeyToItem(UINT nChar, UINT nIndex, HWND hWnd); | | MSG\_OCM\_CHARTOITEM | LRESULT OnCharToItem(UINT nChar, UINT nIndex, HWND hWnd); | | MSG\_OCM\_HSCROLL | void OnHScroll(int nSBCode, short nPos, HWND hWnd); | | MSG\_OCM\_VSCROLL | void OnVScroll(int nSBCode, short nPos, HWND hWnd); | | MSG\_OCM\_CTLCOLOREDIT | LRESULT OnCtlColorEdit(HDC hDC, HWND hWnd); | | MSG\_OCM\_CTLCOLORLISTBOX | LRESULT OnCtlColorListbox(HDC hDC, HWND hWnd); | | MSG\_OCM\_CTLCOLORBTN | LRESULT OnCtlColorBtn(HDC hDC, HWND hWnd); | | MSG\_OCM\_CTLCOLORDLG | LRESULT OnCtlColorDlg(HDC hDC, HWND hWnd); | | MSG\_OCM\_CTLCOLORSCROLLBAR | LRESULT OnCtlColorScrollbar(HDC hDC, HWND hWnd); | | MSG\_OCM\_CTLCOLORSTATIC | LRESULT OnCtlColorStatic(HDC hDC, HWND hWnd); | | MSG\_WM\_CLEAR | void OnClear(); | | MSG\_WM\_COPY | void OnCopy(); | | MSG\_WM\_CUT | void OnCut(); | | MSG\_WM\_PASTE | void OnPaste(); | | MSG\_WM\_UNDO | void OnUndo(); | |