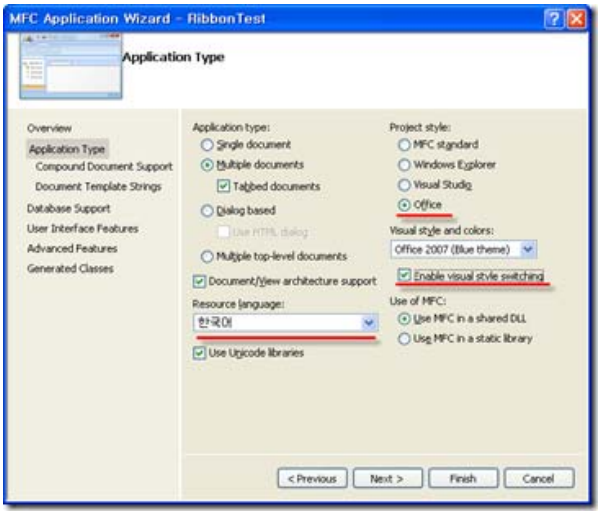
POST 프로그래밍/TIP& Study

[MFC 9.0] Office 스타일의 Ribbon Bar (리본 메뉴) 만드는 법

1. 개요

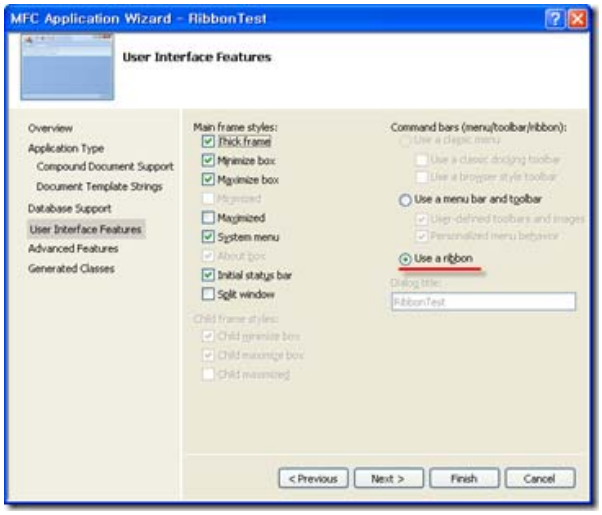
MFC 9.0 Beta 버전에 포함된 Office 2007 스타일의 Ribbon Bar를 만드는 방법을 알아본다.
(Application Wizard가 자동으로 생성하는 코드에 대한 이해를 목적으로 한다.)

2. 프로젝트 생성



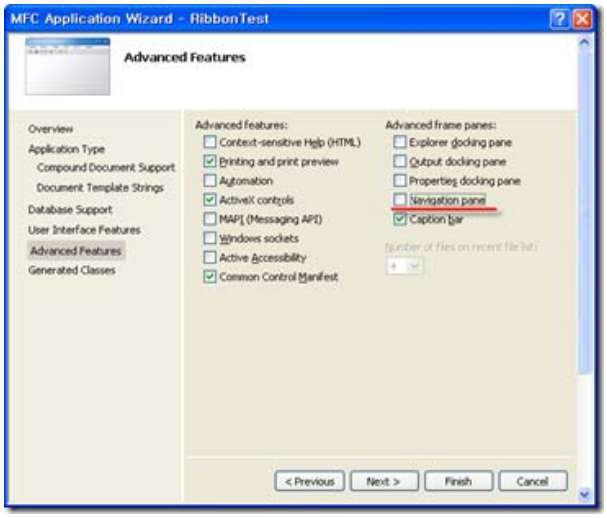
[그림 1 Application Type]

a) [그림 1]과 같이 Project Style을 Office로 선택하고 마음에 드는 Style Thema를 선택한다.
(여기서 주의할 점이 있는데 Enable visual style switching 체크를 해제하게 되면 컴파일 에러가 나게 되는데, 물론 쉽게 에러나는 곳을 고칠 수 있으나 성가신 사항이므로 일단 체크를 하자. 컴파일 에러에 관한 내용은 다음 기회에 포스팅할 예정이다.)



[그림 2 User Interface Features]

b) [그림 2] 와 같이 User Interface Features의 옵션을 선택하는 부분에서 우리는 Ribbon Bar를 사용하여야 하기 때문에 User a ribbon 라디오 버튼에 체크하자. (뭐 기본 선택사항이니 가만히 두어도 된다)



[그림 3 Advanced Features]

c) [그림 3]에 보면 Advanced docking panes 옵션을 보면 5가지를 추가로 선택할 수 있다.
위의 3개의 체크 박스는 Visual Studio Style의 프로젝트를 만들때 흔히 쓰일 듯한 옵션이다.
4번째의 **Navigation pane**은 쉘 폴더 트리뷰와 달력 뷰가 샘플로 들어가는 옵션인데 일단 지금은 Ribbon Bar에 집중하기 위해서 일단 체크를 해제하도록 한다. 5번째의 **Caption bar**는 Ribbon bar 밑에 나오는 녀석으로 사용자에게 간단한 메세지를 출력해주는 기능을 하는 것으로 예를 들면 IE에서 보안설정으로 파업차단, ActiveX 설치시 삑 하고 나타나는 영역이라고 생각하면 되겠다.

d) 여기까지 했다면 Finish 버튼을 클릭하면 프로젝트를 생성할 수 있는데, 바로 빌드를 해보면 [그림 4]와 같이 정말 Office 2007스러운 미려한 디자인의 프로그램을 만날 수 있다.

[그림 4 샘플]

3. Application Wizard가 생성한 코드 이해하기

a) CMainFrame의 OnCreate 이해 하기

MFC 8.0 이하의 버전에서는 MDI 프로젝트에서 CMainFrame 클래스는 CFrameWnd를 상속받았었다. 그러나 리본바를 사용하기 위해서는 CFrameWnd 클래스 대신에 이의 확장 버전인 CMDIFrameWndEx 클래스를 상속받고 있다. 아래의 코드는 Wizard가 자동으로 생성해주는 코드인데 MFC 8.0 이하 버전과의 차이점을 중심으로만 살펴보기로 하겠다.

```
int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CMDIFrameWndEx::OnCreate(lpCreateStruct) == -1)
        return -1;
    // set the visual manager and style based on persisted value
    OnApplicationLook(theApp.m_nAppLook);
    CMDITabInfo mdiTabParams;
    mdiTabParams.m_style = CMFCTabCtrl::STYLE_3D_ONENOTE; // other styles available...
    mdiTabParams.m_bActiveTabCloseButton = TRUE;      // set to FALSE to place close button at right of tab area
    mdiTabParams.m_bTabIcons = FALSE;    // set to TRUE to enable document icons on MDI taba
    mdiTabParams.m_bAutoColor = TRUE;    // set to FALSE to disable auto-coloring of MDI tabs
    mdiTabParams.m_bDocumentMenu = TRUE; // enable the document menu at the right edge of the tab area
    EnableMDITabbedGroups(TRUE, mdiTabParams);
    m_wndRibbonBar.Create(this);
```

```
    InitializeRibbon();
    if (!m_wndStatusBar.Create(this))
    {
        TRACE0("상태 표시줄을 만들지 못했습니다.₩n");
        return -1;      // 만들지 못했습니다.
    }
    CString strTitlePane1;
    CString strTitlePane2;
    strTitlePane1.LoadString(IDS_STATUS_PANE1);
    strTitlePane2.LoadString(IDS_STATUS_PANE2);
    m_wndStatusBar.AddElement(new CMFCRibbonStatusBarPane(ID_STATUSBAR_PANE1, strTitlePane1, TRUE), strTitlePane1);
    m_wndStatusBar.AddExtendedElement(new CMFCRibbonStatusBarPane(ID_STATUSBAR_PANE2, strTitlePane2, TRUE), strTitlePane2);
    // enable Visual Studio 2005 style docking window behavior
    CDockingManager::SetDockingMode(DT_SMART);
    // enable Visual Studio 2005 style docking window auto-hide behavior
    EnableAutoHidePanes(CBRS_ALIGN_ANY);
    // Create a caption bar:
    if (!CreateCaptionBar())
    {
        TRACE0("Failed to create caption bar₩n");
        return -1;      // 만들지 못했습니다.
    }
    // Enable enhanced windows management dialog
    EnableWindowsDialog(ID_WINDOW_MANAGER, IDS_WINDOWS_MANAGER, TRUE);
    return 0;
}
```

- 처음 부분에 등장하는 생소한 함수가 보일 것이 바로 **OnApplicationLook()**라는 함수인데, 이 함수는 바로 Look & Feel을 선택할 수 있게 해주는 CMDIFrameWndEx 의 센스쟁이 멤버 함수이다. 파라미터로 넘어가는 thhApp.m_nAppLook 은 CMainFrame의 생성자에서 대입된다. 아래와 같이 정의된 값을 대입해보면서 스타일이 어떻게 변하는지 직접 느껴보기 바란다.
    - ID_VIEW_APPLOOK_WIN_2000
    - ID_VIEW_APPLOOK_OFF_XP
    - ID_VIEW_APPLOOK_WIN_XP
    - ID_VIEW_APPLOOK_OFF_2003
    - ID_VIEW_APPLOOK_VS_2005
    - ID_VIEW_APPLOOK_OFF_2007_BLUE
    - ID_VIEW_APPLOOK_OFF_2007_BLACK
    - ID_VIEW_APPLOOK_OFF_2007_SILVER
    - ID_VIEW_APPLOOK_OFF_2007_AQUA

[그림 5 예전 MDI 방식]

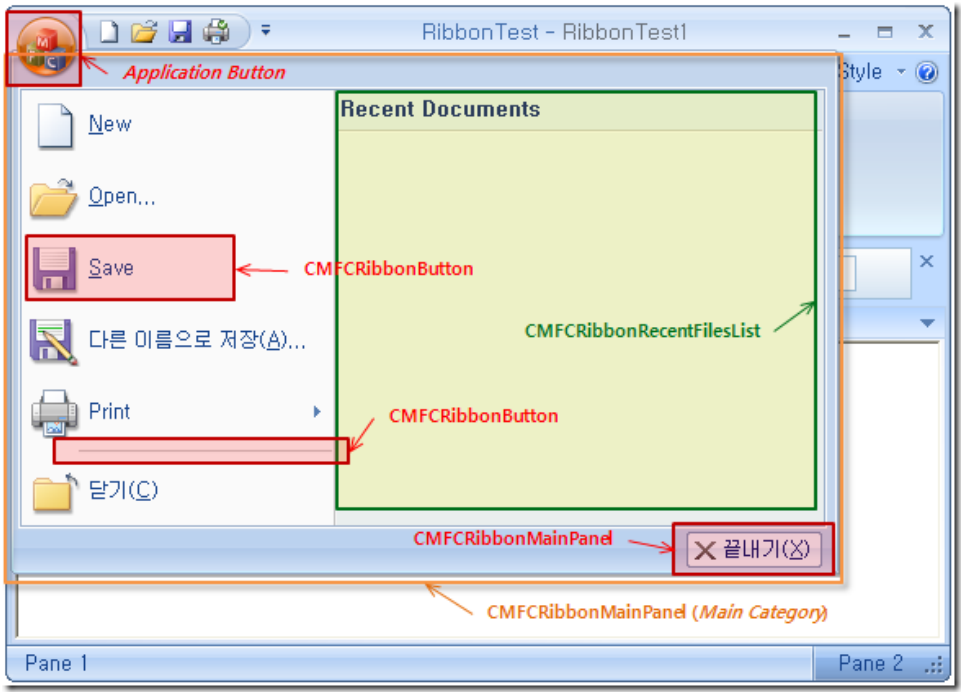
- MDI 형식의 프로젝트이기 때문에 여러개의 문서를 어떤 식으로 표시를 해줄 것인지를 결정하는 함수가 **EnableMDITabbedGroups()** 이다. 첫번째 파라미터를 FALSE로 넘기면 Tab 방식으로 문서를 정렬하는 방식이 아닌 [그림 5]와 같이 예전 MDI 방식으로 사용할 수 있다.
  두번째 파라미터로 넘기는 CMDITabInfo 클래스는 아래와 멤버 변수를 public으로 가지고 있다.

```
class CMDITabInfo
{
public:
    CMDITabInfo();
    void Serialize(CArchive& ar);
    CMFCTabCtrl::Location m_tabLocation;    // Tab에 위치를 아래에 둘지 위의 둘지 선택 (기본 위)
    CMFCTabCtrl::Style    m_style;          // Tab Look & Feel 선택 (아래 참조)
    BOOL m_bTabIcons;                       // Tab에 아이콘을 표시할 것 인지
    BOOL m_bTabCloseButton;                 // Tab에 닫기 버튼을 표시할 것 인지  (m_bActiveTabCloseButton 가 FALSE 일 때)
    BOOL m_bTabCustomTooltips;
    BOOL m_bAutoColor;                      // 새로운 문서가 추가될 때 마다 Tab의 색상이 바뀔것 인지
```

```
    BOOL m_bDocumentMenu;           // 문서가 많이 열렸을 때 좌우 스크롤을 할지 Menu에 표시할 지
    BOOL m_bEnableTabSwap;          // 드래그 앤 드랍으로 탭의 위치를 변경할 수 있게 만들지
    BOOL m_bFlatFrame;              // Frame 의 모습
    BOOL m_bActiveTabCloseButton;   // 활성화된 Tab에 닫기 버튼이 표시되게 할 지 Tab영역 오른쪽 가장자리에  표시할 지
    int  m_nTabBorderSize;          // Frame Border의 굵기
};
```

- CMdlTabInfo의 m_style이라는 멤버 변수는 아래와 같은 값을 가질 수 있다.
  - STYLE_3D
  - STYLE_FLAT
  - STYLE_FLAT_SHARED_HORZ_SCROLL
  - STYLE_3D_SCROLLED
  - STYLE_3D_ONENOTE
  - STYLE_3D_VS2005
  - STYLE_3D_ROUNDED
  - STYLE_3D_ROUNDED_SCROLL
- InitializeRibbon() 함수에 리본바를 구성하는 모든 코드들이 속해있다. (기본적으로 Wizard가 샘플로 만들어 놓은 함수이다)

b) Main Category 부분 구성하기



[그림 6 Main Category의 구성]

MFC 9.0과 같이 나온 미려한(?) 디자인의 MFC이 아이콘이 속해있는 버튼이 보이는가? 그 버튼을 누르면 위의 [그림 6]처럼 Main Category라는 익숙한 창이 나타나게 되는데 일단 MainCategory를 뛰워주게 하는 **이 Application Button이라고 부르는 녀석을 먼저 살펴보도록 한다.**

CMainFrame는 멤버 변수로 **CMFCRibbonApplicationButton** class의 객체를 **m_MainButton**라는 이름으로 가지고 있다. 이 버튼의 이미지는 Bitmap파일의 리소스 아이디를 SetImage함수를 통해 전달해 주면 된다.

그리고 Alt 키를 누르면 각각의 Ribbon Element들이 가지는 단축키들이 표시되는데, 이 단축키를 세팅해주는 방식 두가지가 있다.

```
m_MainButton.SetText(_T("₩nf"));  // Element의 이름과 단축키를 동시에 설정하는 방법
m_MainButton.SetKeys(_T("f"));    // 단축키 값만을 설정하는 방법
```

그런데 한가지 의문점이 든다. SetText의 문자열이 개행문자로 시작하는 부분이다.
CMFCRibbonApplicationButton의 멤버함수(정확히는 CMFCRibbonBaseElement의 멤버함수) SetText 함수는 단축키와 표시될 이름을 동시에 설정하는 함수이다. 단, 위와 같이 '₩n' 개행문자를 구분자로 표시될 이름과 단축키가 한 문자열로 전달되어야 한다. m_MainButton에 SetText를 하는 것은 약간 이례적인 경우인데, Application Button(m_MainButton)은 [그림 6]을 보다시피 문자열로 표현되는 부분이 없다. 그래서 SetText함수를 사용하여

```
m_MainButton.SetText(_T("MainButton!₩nf"));
```

위와 같이 코딩하여도 '₩n' 앞의 문자열 MainButton은 쓰여질 곳은 없기 때문에 **굳이 넣지 않아도 된다.**
이제 의문이 해소되었는가?

아직도 이상하다고 생각되면 그냥 SetKeys함수를 사용하여 그 의미를 명확하게 하자.

참고로 사실 Application Wizard가 생성해주는 코드에 저 부분이 들어있어서 설명해주고 싶었다.
ApplicationButton의 경우 이상하다고 느낄지는 모르겠지만 예를 들어 다른 Ribbon Element 들을 만들때 생성자로

```
CMFCRibbonButton* pBtnPaste = new CMFCRibbonButton(ID_EDIT_PASTE, _T("Paste₩nV"), 0, 0);
```

와 같이 두번째 파라미터로 문자열 값을 넣었을 때 생성자 내부에서 CMFCRibbonBaseElement의 SetText함수를 호출하게 되어 있어서 그 Element는 Paste라는 이름을 가지게 되고 단축키는 V를 가지게 된다.
주의할 점은 소문자를 넣어도 Alt를 눌렀을 때의 모든 단축키의 모습은 아래 그림과 같이 대문자로 나온다는 것이다.

[그림 7 단축키 표현 방식]

마지막으로 버튼이라면 마우스가 오버 했을 때 툴팁을 뛰워주야하지 않겠는가? 그래서 그 툴팁 텍스트를 설정하는 부분이 바로

```
m_MainButton.SetToolTipText(_T("File"));
```

SetToolTipText 함수를 사용하는 부분이다.

이렇게 하면 Button에 대한 속성은 거의 다 정해준 것 같다.
그러나 여기까지는 그냥 버튼을 하나 생성시켜서 메모리 어느 공간에 있을 뿐이다.
이 버튼이 ApplicationButton이다!! 라고 설정을 시켜주어야 한다.

OnCreate 함수에서 Create 시켜주었던 RibbonBar 객체가 생각나는가? 그 RibbonBar에다 이 버튼이 Application이고 버튼 크기는 얼마였으면 좋겠다라고 등록시켜주는 함수가 있다.

```
m_wndRibbonBar.SetApplicationButton(&m_MainButton, CSize (45, 45)); //가로 45px, 세로 45px 크기로 ApplicationButton으로 등록
```

위의 코드처럼 **SetApplicationButton**이라는 함수인데, 만약에 위와 같이 m_MainButton을 RibbonBar에 저 함수를 통해 등록하지 않으면 어떻게 될까?

[그림 8 Application Button이 없는 리본바]

[그림 8] 처럼 ApplicationButton이 없는 UI가 나타난다.
경우에 따라 ApplicationButton이 없는 경우를 선호할 때는 CMFCRibbonApplicationButton 을 설정하고 RibbonBar에 등록하는 부분을 주석처리하면 그 뿐인 것이다.

길게 길게 ApplicationButton에 대해서 설명을 했다.
그렇다면 이번에는 [그림 6] 처럼 **MainCategory에 New Open 처럼 항목을 집어 넣는 부분을 살펴보자.**
일단 MainCategory는 좀 특이하다.
RibbonBar에 항상 나타나 있는 것이 아니라 ApplicationButton을 통해서 나타나는 것도 그렇고, 기본적으로 MDI 프로그램에 꼭 필요한 항목들만 들어 있다.
그래서인지 RibbonBar에서는 AddMainCategory 함수가 따로 존재한다.
(뒤에 보면 알겠지만 그냥 AddCategory 함수가 있다. AddCategory는 여러 카테고리를 추가할 수 있지만, AddMainCategory는 당연히 한번만 호출 되어서 하나의 MainCategory만 존재해야 한다.)

```
CMFCRibbonMainPanel* AddMainCategory(
  LPCTSTR lpszName,                  // 메인카테고리의 이름
  UINT uiSmallImagesResID,           // 작은 이미지 리소스 아이디
  UINT uiLargeImagesResID,           // 큰 이미지 리소스 아이디
  CSize sizeSmallImage=CSize(16, 16) ,  // 작은 이미지 하나의 크기
  CSize sizeLargeImage=CSize(32, 32)   // 큰 이미지 하나의 크기
);
```

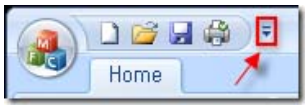
너무나 당연한 이야기지만, 리소스에 포함된 이미지는 파라미터로 넘겨주는 크기의 이미지 조각을 연속적으로 붙여놓은 하나의 이미지여야한다.

```
CMFCRibbonMainPanel* pMainPanel = m_wndRibbonBar.AddMainCategory(_T("File"), IDB_FILESMALL, IDB_FILELARGE);
```

그런데 여기서 질문.

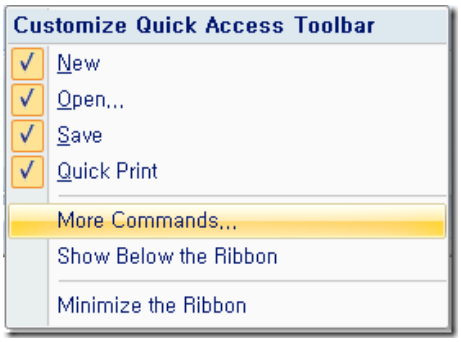
1. 메인 카테고리 이름이라고 준 File 이라는 문자열이 Application을 눌러봐도 나오는 부분이 없는 것 같다
2. MainCategory창에는 전부 큰 이미지(Large Icon)만 보이는데 굳이 작은 이미지를 넣을 필요가 있는가?

차근 차근 위의 질문에 대한 답을 찾아보자.
정답은 QAT(**Quick Access Toolbar**)의 지원때문이라고 간단히 말할 수도 있겠다.



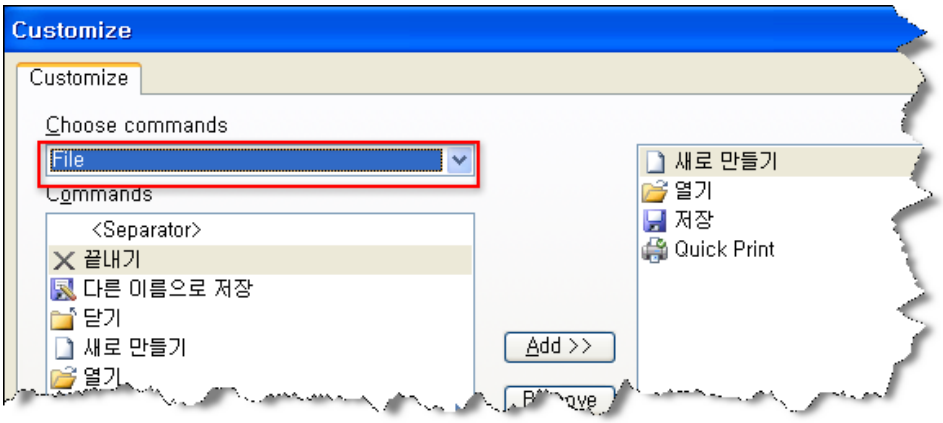
[그림 9 QAT]

Office UX에서 사용자가 가장 자주 쓰는 기능을 ApplicationButton 옆에 보이는 QAT로 모아서 편리하게 쓸 수 있게끔 하는 것이 있다.
그 옆에 아래로 향하는 화살표 부분을 클릭하면 QAT를 Customize할 수 있는 메뉴가 나타난다.



[그림 10 QAT 팝업메뉴]

[그림 10]을 보면 More Commands라는 메뉴가 보이는데 이 항목을 선택했을 때 아래 [그림 11]과 같이 Customize 할 수 있는 창이 뜬다.



[그림 10] QAT Customize 창

Choose commands 부분을 보면 AddMainCategory에서 이름으로 넘겨주었던 File 이라는 문자열을 발견할 수 있다.(질문 1에 대한 답)
또한 RibbonBar에서 보이는 모든 Element들(버튼, 패널, 메뉴 등등)이 모두 QAT에 들어갈 수 있다. 그리고 QAT에서는 그 공간적인 제약때문에 작은 이미지 크기만 표현해줄 수 있다. 그래서 비록 MainCategory에는 큰 이미지만 필요하지만 QAT를 위해 작은 이미지도 필요한 것이다. 또한 Category 안의 Panel 크기 때문에 작은 이미지로 표현 될 때도 있다. (질문 2에 대한 답)

어쨋든 AddMainCategory함수를 통해서 CMFCRibbonMainPanel의 포인터를 return 값으로 받았다.

```
pMainPanel->Add(new CMFCRibbonButton(ID_FILE_NEW, _T("&New"), 0, 0));
pMainPanel->Add(new CMFCRibbonButton(ID_FILE_OPEN, _T("&Open..."), 1, 1));
pMainPanel->Add(new CMFCRibbonButton(ID_FILE_SAVE, _T("&Save..."), 2, 2));
pMainPanel->Add(new CMFCRibbonButton(ID_FILE_SAVE_AS, _T("Save &as"), 3, 3));
```

CMFCRibbonMainPanel의 포인터 pMainPanel의 Add 함수를 통해 CMFCRibbonButton을 추가할 수 있다.
근데 Add함수안의 파라미터를 바로 new 해서 버튼 객체를 넘겨주는게 약간 미심쩍은 개발자들이 있을 것이다.
(사실 unmanaged code를 작성하는 c/c++ 개발자들은 근본적으로 불안해야할 코드처럼 보일 것이다.)

그러나 CMFCRibbonBar와 CMFCRibbonCategory 소멸자에는 아래와 같은 코드가 있다.

```
CMFCRibbonBar::~CMFCRibbonBar()
{
```

```
    int i = 0;
    for (i = 0; i < m_arCategories.GetSize(); i++)
    {
        ASSERT_VALID(m_arCategories [i]);
        delete m_arCategories [i];
    }
    for (i = 0; i < m_arContextCaptions.GetSize(); i++)
    {
        ASSERT_VALID(m_arContextCaptions [i]);
        delete m_arContextCaptions [i];
    }
    if (m_pMainCategory != NULL)
    {
        ASSERT_VALID(m_pMainCategory);
        delete m_pMainCategory;
    }
}
CMFCRibbonCategory::~CMFCRibbonCategory()
{
    int i = 0;
    for (i = 0; i < m_arPanels.GetSize(); i++)
    {
        delete m_arPanels [i];
    }
    for (i = 0; i < m_arElements.GetSize(); i++)
    {
        delete m_arElements [i];
    }
}
```

즉, m_wndRibbonBar가 소멸되는 시점에서 모든 카테고리 객체가 해제되고, 카테고리 객체가 해제될 때 카테고리가 가지고 있던 Panel과 Element 객체를 해제시키는 것이다. 그렇기 때문에 new 키워드에 대한 불안감은 접어도 될 듯하다.

참고로 CMFCRibbonButton은 아래와 같이 2개의 생성자가 오버로딩 되어있다.

```
CMFCRibbonButton(
    UINT nID,                       // Button의 Command Id
    LPCTSTR lpszText,               // Button의 Text Label
    int nSmallImageIndex=-1,        // AddMainCategory에서 지정한 작은 이미지의 인덱스
    int nLargeImageIndex=-1,        // AddMainCategory에서 지정한 큰 이미지의 인덱스
    BOOL bAlwaysShowDescription=FALSE  // 항상 설명(Description) 부분을 표시할 지 선택.
);
CMFCRibbonButton(
    UINT nID,
    LPCTSTR lpszText,
    HICON hIcon,                                // 큰 이미지 아이콘
    BOOL bAlwaysShowDescription=FALSE,    // 항상 설명 부분을 표시할 지 선택
    HICON hIconSmall=NULL,                      // 작은 이미지 아이콘
    BOOL bAutoDestroyIcon=FALSE,                // Button 객체가 사라질 때 소멸자가 DestroyIcon 를 부를지 선택
    BOOL bAlphaBlendIcon=FALSE                  // 아이콘이 알파 블렌딩 아이콘인지 선택
);
```

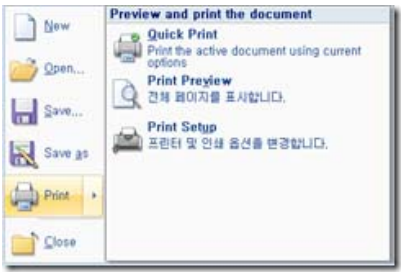
아래 코드는 Print 버튼을 만들고 버튼의 서브 아이템을 추가하는 코드이다.

```
CMFCRibbonButton* pBtnPrint = new CMFCRibbonButton(ID_FILE_PRINT, _T("Print"), 6, 6);
        pBtnPrint->SetKeys(_T("p"), _T("w"));
        pBtnPrint->AddSubItem(new CMFCRibbonLabel(_T("Preview and print the document")));
        pBtnPrint->AddSubItem(new CMFCRibbonButton(ID_FILE_PRINT_DIRECT, _T("&Quick Print"), 7, 7, TRUE));
        pBtnPrint->AddSubItem(new CMFCRibbonButton(ID_FILE_PRINT_PREVIEW, _T("Print Pre&view"), 8, 8, TRUE));
        pBtnPrint->AddSubItem(new CMFCRibbonButton(ID_FILE_PRINT_SETUP, _T("Print Set&up"), 11, 11, TRUE));
    pMainPanel->Add(pBtnPrint);
```

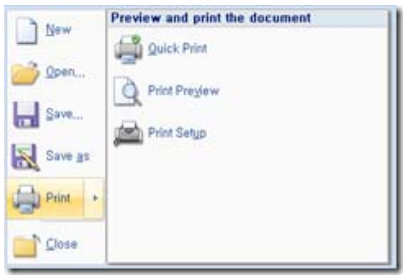
SetKeys라는 함수는 ApplicationButton의 단축키를 설정할 때 잠깐 보았던 함수이다.
그러나 여기서 진정한 SetKeys함수가 빛을 발한다. 두번째 인자로 주는 문자열은 어디에 쓰는 단축키일까?
바로 Print 버튼에 있는 하위 아이템의 묶음을 오픈하는 단축키이다.

물론 하위 메뉴가 없을 때는 아무런 소용이 없을 것이다.

AddSubItem으로 CMFCRibbonLabel과 CMFCRibbonButton을 추가하는 부분이 있는데, CMFCRibbonBaseElement를 상속받은 Class의 객체는 원하는 수만큼 추가를 할 수 있다. (흡사 WPF의 StackPanel과 같이 생겼다.) 너무 많은 아이템들이 추가되어 모두 화면에 표시되지 않을 때는 자동으로 스크롤 할 수 있는 화살표 버튼도 자동으로 추가된다.



[그림 11, bAlwaysShowDescription = TRUE]



[그림 12, bAlwaysShowDescription = FALSE]

[그림 11]과 [그림 12]의 차이점이 보이는가?
저것이 바로 CMFCRibbonButton을 생성할 때 파라미터로 주는 bAlwaysShowDescription의 속성이다.
이제 버튼의 하위메뉴를 생성하는 방법도 알았으니 거의 마무리 단계이다.

```
pMainPanel->Add(new CMFCRibbonSeparator(TRUE));
pMainPanel->Add(new CMFCRibbonButton(ID_FILE_CLOSE, _T("&Close"), 9, 9));
pMainPanel->AddRecentFilesList(_T("Recent Documents"));
pMainPanel->AddToBottom(new CMFCRibbonMainPanelButton(ID_APP_EXIT, _T("E&xit"), 15));
```

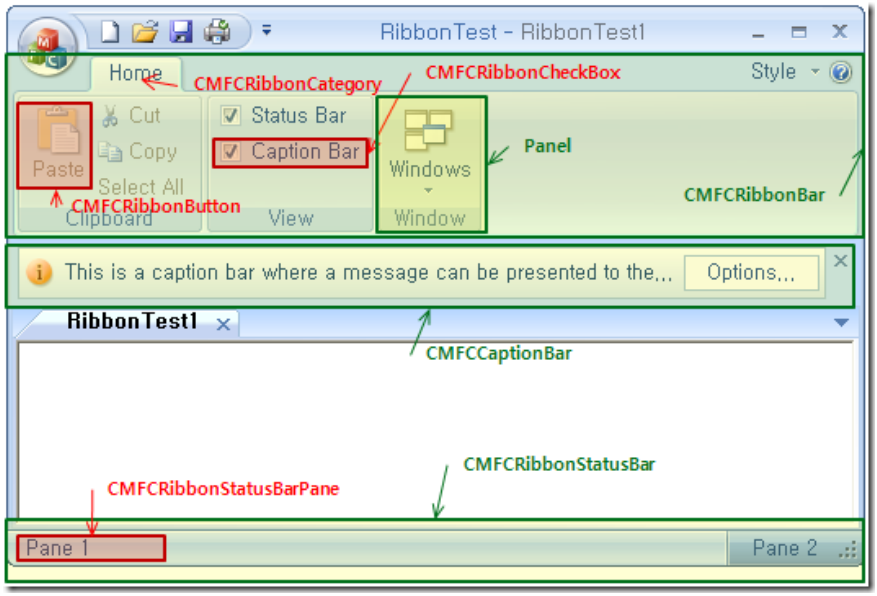
CMFCRibbonSeparator 는 생성자에 TRUE를 주면 가로 구분자가, FALSE를 주면 세로 구분자가 생긴다.
마지막으로 중요한 AddRecentFilesList 함수가 있는데 두번째 파라미터로 Width 값을 줄 수있다. 그 너비에 따라 MainCategory의 전체크기가 바뀌게 된다.
아래의 [그림 13]을 보면 Print 버튼의 하위 메뉴 크기도 덩달아 조정되는 것을 알 수 있다.
참고로 AddRecentFilesList 함수는 내부적으로 AddToRight(new CMFCRibbonRecentFilesList(lpszLabel), nWidth) 와 같은 부분이 있다.

[그림 13, RecentFilesList가 없을 경우 MainCategory]

끝으로 CMFCRibbonMainPanel 을 AddToBottom 함수로 추가하고 있는 부분이 있는데 UI 성격상 임의로 작은 버튼만 들어가게 해놓은 것 같다. AddToBottom 함수에는 그냥 CMFCRibbonButton을 쓸 수 없고 CMFCRibbonButton을 상속받은 CMFCMainPanelButton class의 객체만 받아들인다는 것에 주의하고, 작은 이미지 리스트의 인덱스를 넘겨주어야 한다는 것을 명심하면 되겠다.

c) 본격적인 RibbonBar 구성하기

앞에서 ApplicationButton, MainCategory를 구성하는 방법을 알아보았다면 이제 정말 RibbonBar를 디자인하는 방법을 알아보겠다.



[그림 14, RibbonBar 구성]

RibbonBar를 구성할 때는 크게 3가지 과정을 거친다.

1. **CMFCRibbonBar의 객체 m_wndRibbonBar의 멤버함수 AddCategory를 이용해 Category를 만든다.**
   (AddCategory 의 return type은 CMFCRibbonCategory의 포인터이다.)
2. **CMFCRibbonCategory의 멤버함수 AddPanel를 이용해 Panel을 만든다.**
   (AddPanel의 return type은 CMFCRibbonPanel의 포인터이다.)
3. CMFCRibbonPanel의 멤버함수 Add를 이용하여 CMFCRibbonBaseElement를 상속받은 RibbonElement들을 원하는 만큼 추가한다.
   (단, RibbonElement는 독립적인 CommandID를 부여하여야 하고 경우에 따라 Handler를 만들어야 한다)

Application Wizard는 [그림 14]와 같이 RibbonBar에 Home Category를 추가하였고 이 Category에 3개의 Panel을 추가하였다.
Clipboard라는 Panel에는 4개의 CMFCRibbonButton을 가지고
View라는 Panel에는 2개의 CMFCRibbonCheckBox를 가지고
Windows라는 Panel에는 1개의 CMFCRibbonButton을 가지는데 그 버튼은 하위메뉴를 포함하고 있다.

```
// Add "Home" category with "Clipboard" panel:
CMFCRibbonCategory* pCategoryHome = m_wndRibbonBar.AddCategory(_T("&Home"), IDB_WRITESMALL, IDB_WRITELARGE);
    // Create "Clipboard" panel:
    CMFCRibbonPanel* pPanelClipboard = pCategoryHome->AddPanel(_T("Clipboard₩nd"), m_PanelImages.ExtractIcon(27));
        CMFCRibbonButton* pBtnPaste = new CMFCRibbonButton(ID_EDIT_PASTE, _T("Paste₩nv"), 0, 0);
        pPanelClipboard->Add(pBtnPaste);
        pPanelClipboard->Add(new CMFCRibbonButton(ID_EDIT_CUT, _T("Cut₩nx"), 1, 1));
        pPanelClipboard->Add(new CMFCRibbonButton(ID_EDIT_COPY, _T("Copy₩nc"), 2));
        pPanelClipboard->Add(new CMFCRibbonButton(ID_EDIT_SELECT_ALL, _T("Select All₩na"), -1));

    // Create and add a "View" panel:
    CMFCRibbonPanel* pPanelView = pCategoryHome->AddPanel(_T("View"), m_PanelImages.ExtractIcon (7));
        CMFCRibbonButton* pBtnStatusBar = new CMFCRibbonCheckBox(ID_VIEW_STATUS_BAR, _T("Status Bar"));
        pPanelView->Add(pBtnStatusBar);
        CMFCRibbonButton* pBtnCaptionBar = new CMFCRibbonCheckBox(ID_VIEW_CAPTION_BAR, _T("Caption Bar"));
        pPanelView->Add(pBtnCaptionBar);

    // Create and add a "Windows" panel:
    CMFCRibbonPanel* pPanelWindow = pCategoryHome->AddPanel(_T("Window₩nw"), m_PanelImages.ExtractIcon (7));
        CMFCRibbonButton* pBtnWindows = new CMFCRibbonButton(ID_WINDOW_MANAGER, _T("Windows₩ni"), -1, 1);
            pBtnWindows->SetMenu(IDR_WINDOWS_MENU, TRUE);
        pPanelWindow->Add(pBtnWindows);
```

위의 코드를 살펴보면 빨간색으로 표시된 부분이 Category를 만들고, 그 Category에 Panel을 추가하고, 또 그 Panel에 RibbonElement 들을 추가하는 부분이다. AddCategory는 위에서 본 AddMainCategory와 유사한 형식으로 사용하면 된다. AddPanel은 표시될 이름₩n단축키 문자열로 첫번째 파라미터를 넘겨주고, QAT(Quick Access Toolbar)에 표시될 때 사용할 작은 이미지 아이콘 핸드를 넘겨주면된다. (물론 생략가능하다.)

마지막으로 메세지 핸들러를 만들어 주면 그 Ribbon Element가 클릭되었을 때 원하는 함수를 호출할 수 있다.

```
BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWndEx)
```

```
    ON_WM_CREATE()
    ON_COMMAND(ID_WINDOW_MANAGER, &CMainFrame::OnWindowManager)
    ON_COMMAND_RANGE(ID_VIEW_APPLOOK_WIN_2000, ID_VIEW_APPLOOK_OFF_2007_AQUA, OnApplicationLook)
   ON_UPDATE_COMMAND_UI_RANGE(ID_VIEW_APPLOOK_WIN_2000, ID_VIEW_APPLOOK_OFF_2007_AQUA, OnUpdateApplicationLook)
    ON_COMMAND(ID_VIEW_CAPTION_BAR, OnViewCaptionBar)
    ON_UPDATE_COMMAND_UI(ID_VIEW_CAPTION_BAR, OnUpdateViewCaptionBar)
    ON_COMMAND(ID_FILE_NEW, &CMainFrame::OnFileNew)
    ON_COMMAND(ID_FILE_CLOSE, &CMainFrame::OnFileClose)
END_MESSAGE_MAP()
```

예를 들어 ID_VIEW_CAPTION_BAR의 메세지 핸들러를 만들고 싶다면 afx_msg void OnViewCaptionBar(); 와 같이 afx_msg 를 앞에 붙여서 함수를 선언한다
음 위와 같이 메세지 맵에 ON_COMMAND(ID_VIEW_CAPTION_BAR, OnViewCaptionBar) 를 추가시켜주면 된다.
물론 OnViewCaptionBar 함수는 원하는 데로 구현하면 되는 것이다.
만약 Command ID가 연속적이고 각각의 메세지를 한 핸들러로 등록하고 싶다면 ON_COMMAND 대신에 ON_COMMAND_RANGE 매크로를 사용하면 된다.

d) QAT(Quick Access Toolbar)

```
CList<UINT, UINT> lstQATCmds;
lstQATCmds.AddTail(ID_FILE_NEW);
lstQATCmds.AddTail(ID_FILE_OPEN);
lstQATCmds.AddTail(ID_FILE_SAVE);
lstQATCmds.AddTail(ID_FILE_PRINT_DIRECT);
m_wndRibbonBar.SetQuickAccessCommands(lstQATCmds);
```

기본적으로 표시될 QAT 목록은 위와 같이 CList로 CommandId를 구성하여 SetQuickAccessCommands 함수를 통해 넘겨주면 간단하게 할 수 있다.

4. 주의점

- 위의 설명중 사용된 코드는 가독성을 위해 Application Wizard가 생성한 코드를 약간 수정한 부분이 있다.
  예를 들면 StringTable에서 문자열을 가져오기 위해 CString의 LoadString 함수를 사용하는 부분은 생략하였다.
- QAT나 RibbonBar를 최소화 시키는 기능들은 사용자가 선택하는 기능이고 Customize가 가능한 기능이다.
  이러한 정보는 Registry HKEY_CURRENT_USER₩[App에서 설정한 레지스트리 키]₩[프로그램 이름]₩Workspace에서 확인할 수 있다.

chaoskcuf's lab

TOP

posted at
2008/02/24 18:32

FEEDBACK 아직 트랙백이 없음 :: 댓글 2개

트랙백

트랙백 주소 :: http://chaoskcuf.com/trackback/129

댓글

파이어준 2008/02/25 03:18 (링크주소 :: 수정/삭제 :: 답변달기)
와 멋져요! 죽여요! 작살이에요!

chaoskcuf 2008/02/25 11:54 (링크주소 :: 수정/삭제)
이번에 MS가 전략적으로 MFC를 상당부분 추가시켜주었습니다~
MFC 좀더 보고 ASP.NET AJAX 분석해야겠어요.. ^^;;

댓글 달기

◀ recent : [1] : ... [47] : [48] : [49] : [50] : [51] : [52] : [53] : [54] : [55] : ... [162] : previous ▶