Daten und Views

Ziel, Inhalt

- Wir lernen eine Typensichere Array Klasse der MFC kennen und wenden diese in unserem letzten Projekt an
- Wir lernen die Darstellung von Noten

Daten und Views

Typensichere Array Klasse der MFC

CArray

Die MFC bietet seit einiger Zeit Klassen an, die ähnlich wie in der STL mittels Templates für verschiedene Datentypen anwendbar sind. Es gibt eine Klasse CArray:

```
CArray<class TYPE, class ARG_TYPE>;

CArray<int, int> TestArray;
```

Das erste Template Argument gibt an, welcher Datentyp im Array gespeichert wird, währenddem das zweite Argument angibt, was die Zugriffsfunktionen für einen Datentypen haben.

CList

Ebenso bietet die MFC eine Template Klasse CList an. Eine Liste hat gegenüber einem Array den allgemeinen Vorteil, dass das Anfügen eines Elementes immer gleich lang dauert.

```
CList<class TYPE, class ARG_TYPE>;

CList<double, double> DoubleList;
```

CTypedPtrArray

In der Anwendung sehr ähnlich wie ein CObArray ist die Klasse CTypedPtrArray. Im Grunde genommen wird die CObArray Klasse verwendet um die CTypedPtrArray Klasse zu implementieren. Bei der Anwendung sieht man das sehr gut.

```
CTypedPtrArray<CObArray, CNote*>
```

Diese Klasse würde ich anwenden um Zeiger auf beliebige Datentypen zu speichern, wie es in unserem Beispiel von letzter Lektion http://www.devmentor.ch/teaching/additional/00I/Semester6/index.html nötig wäre.

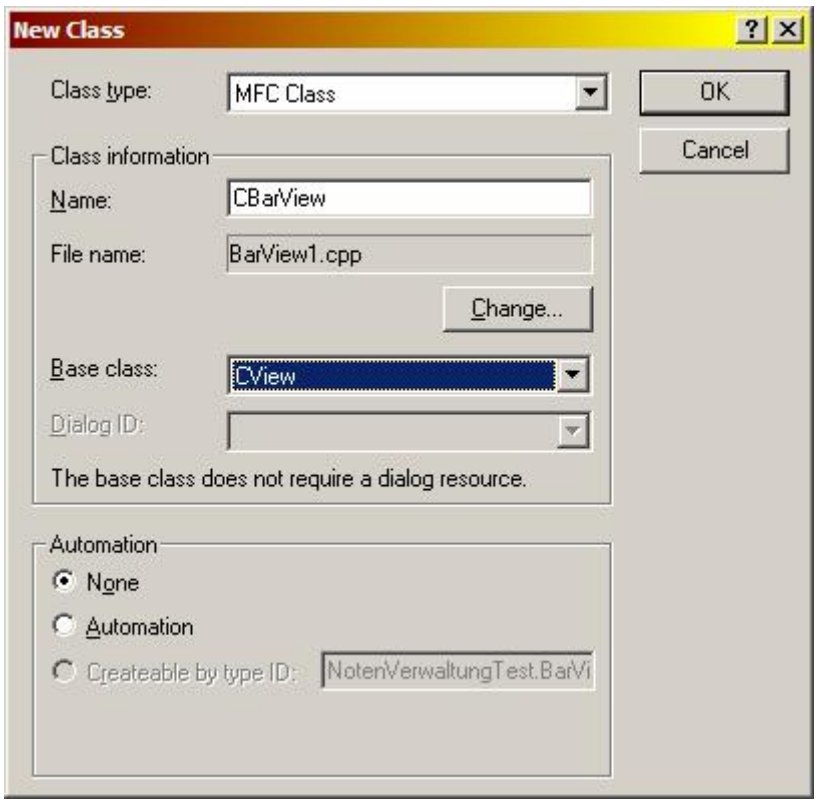
Übung

An dieser Stelle nehmen wir das Projekt von letzter Woche und bauen es um, so dass es anstelle des CObArray ein CTypedPtrArray verwendet.

Views

Neue View erzeugen

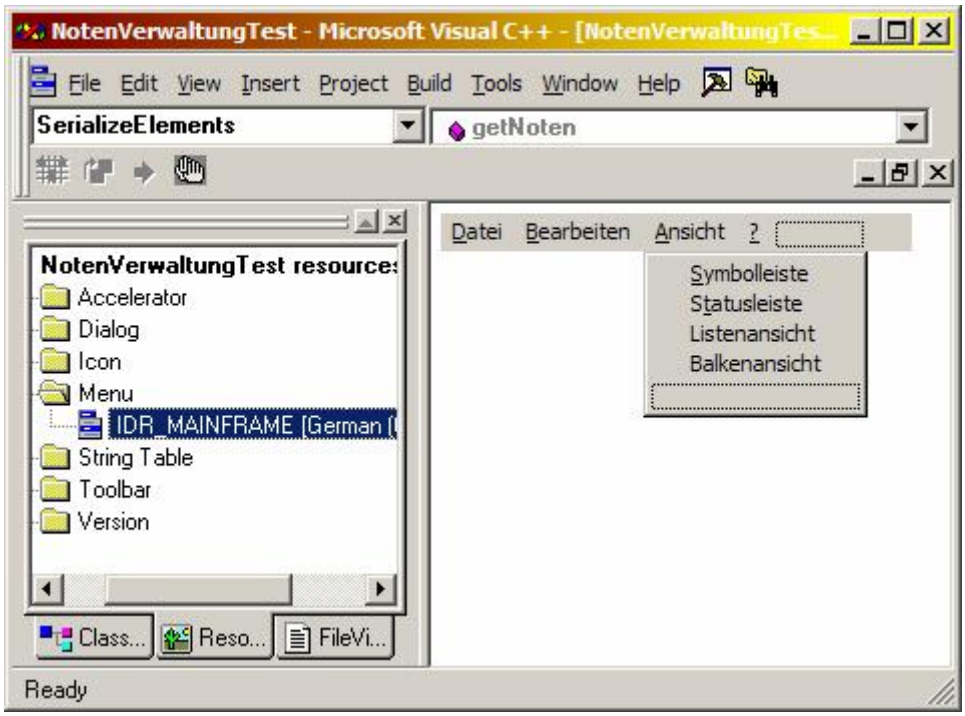
Mit dem Klassenassistenten erzeugen wir eine neue MFC - Klasse, die von CView ableitet.



Klassenassistent zum Erzeugen der neuen View Klasse

View umschalten

Um die Views umzuschalten werden wir zwei Menupunkte hinzufügen. Im Ressourceneditor sehen wir, wenn wir das Menu editieren, den Menupunkt Ansicht, wo wir zwei Menupunkte hinzufügen.

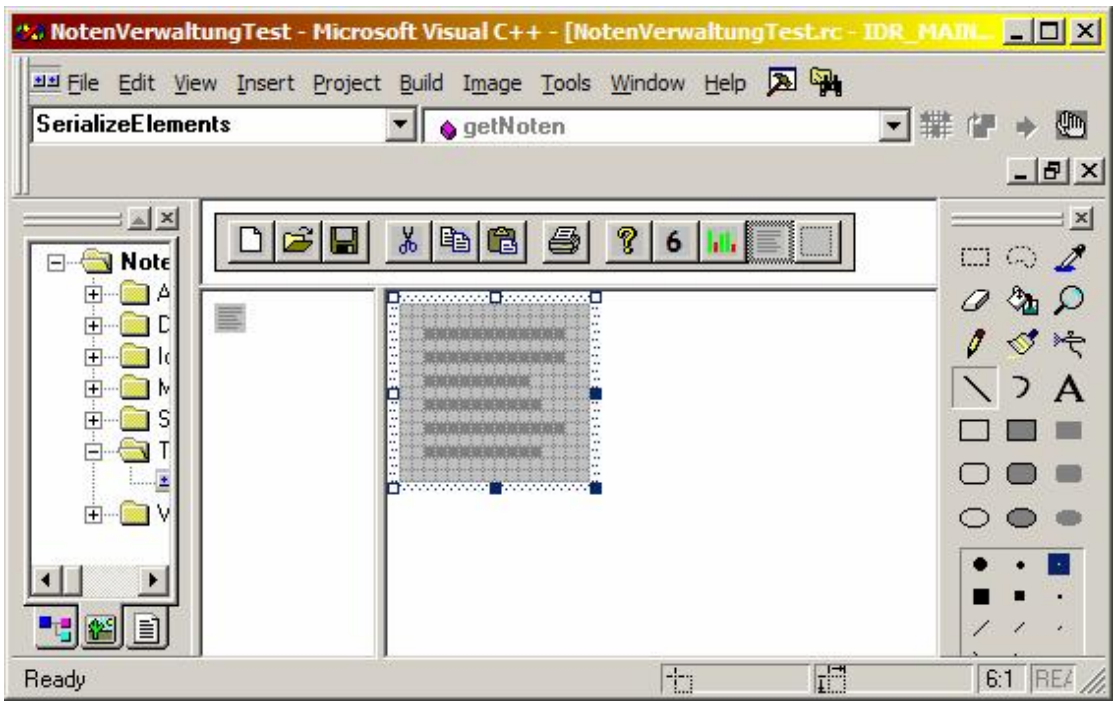


Editieren der Menuressource

Wir müssen unseren beiden Menupunkten eine Menu ID geben und ergänzen auch gleich den Text für den Tooltip:

Hier der Menupunkt Listenansicht

Es ist jetzt auch einfach einen Toolbarbutton zu erzeugen.



Toolbar Buttons erzeugen

Wichtig ist es diesen Toolbarbuttons die gleiche ID zu geben wie dem Menupunkt.

Die Eigenschaften für den Listview Tollbar Button

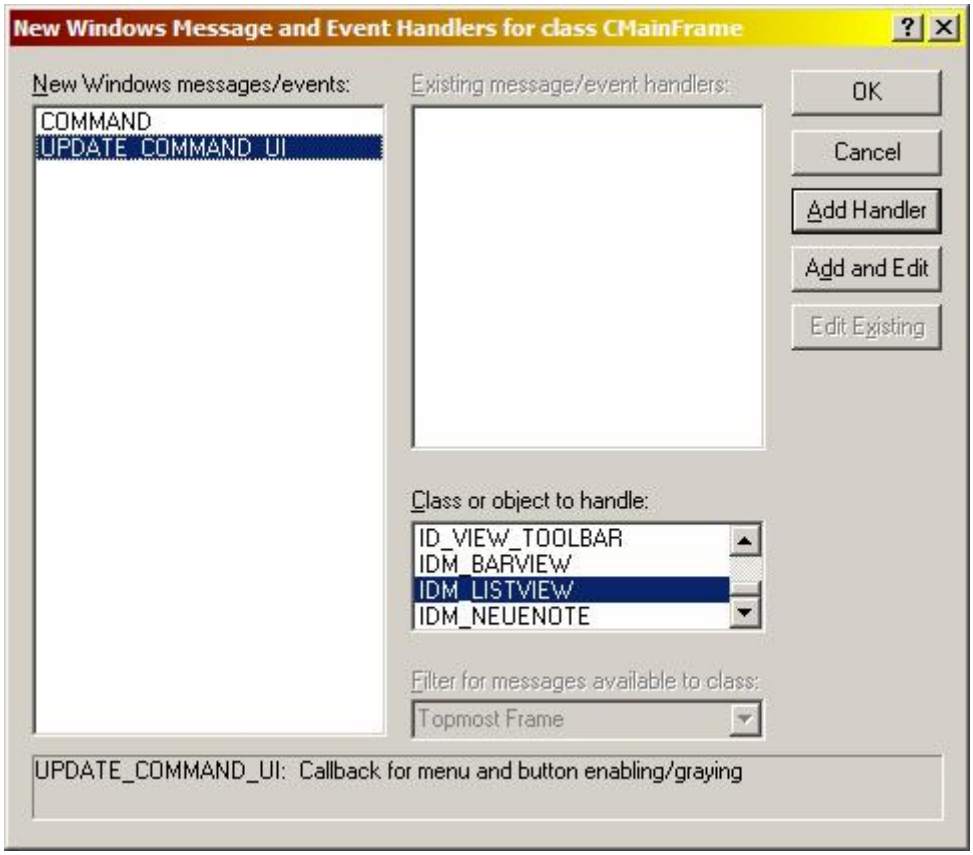
Die Checked Eigenschaft von Commands

Bei solchen Kommandos und Menupunkten wird häufig ein Haken √ verwendet, um anzuzeigen, das etwas aktiv ist. Es gibt in der MFC einen Mechanismus, um dem Framework zu sagen, ob etwas abgehakt sein soll.

CCmdUI

Die Klasse CCmdUI tritt dabei als Helferklasse in Erscheinung. Wir brauchen aber in erster Linie den Klassenassistenten um einen Nachrichtenbehandler einzufügen. Die Entscheidung welche View angezeigt werden soll treffen wir am besten in der Klasse CMainFrame, denn diese beinhaltet die Views auf

dem Bildschirm. Wähle also mit der rechten Maustaste den Eintrag „Nachrichtenbehandler einfügen“ wenn du damit auf die CMainFrame Klasse in der Klassenansicht klickst.
Wähle im Dialog rechts unten die ID, die du dem Listview Menupunkt zugeordnet hast. Als erstes wählen wir diesmal im linken Feld den Eintrag „UPDATE_COMMAND_UI“. Den anderen brauchen wir erst später.



Hier wird der UI – Behandler gewählt

Der Code, der erzeugt wird sieht etwa so aus:

```
void CMainFrame::OnUpdateListview(CCmdUI* pCmdUI)
{
      // TODO: Add your command update UI handler code here

}
```

Hier taucht also ein Objekt der Klasse CCmdUI auf. Jetzt können wir auf diesem Objekt ganz einfach einige Methoden aufrufen. Hier ein Beispiel, das wir testen können:

```
void CMainFrame::OnUpdateListview(CCmdUI* pCmdUI)
{
    pCmdUI->SetCheck();
}
```

Für die View – Umschaltung verwenden wir eine Variable von einem enum, die wir in der CMainFrame Klasse im private Teil einfügen (das geht am besten von Hand, also ohne Klassenassistent):

```
enum ViewType
{
    Listview,
    Barview
};

ViewType m_viewType;
```

Diese Variable initialisieren wir selbstverständlich im Konstruktor. Im Moment wird ja sowieso die Listview als erste View verwendet. Hier der Konstruktor:

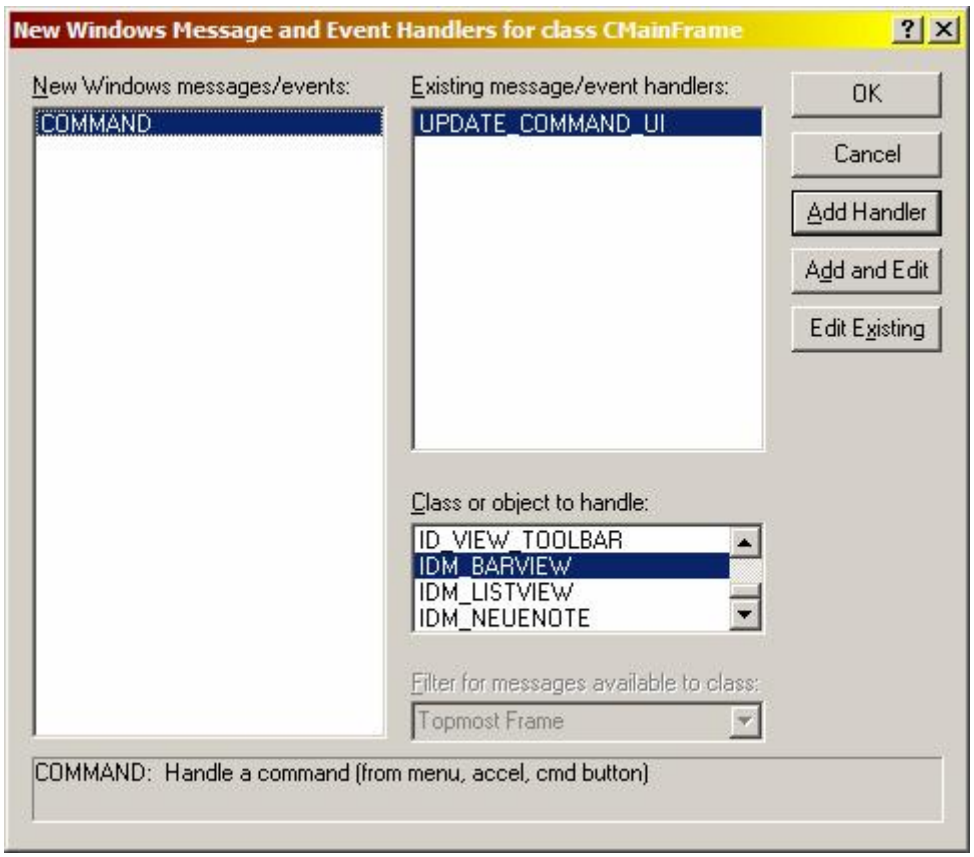
```
//////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()
           :m_viewType(Listview)
{
      // TODO: add member initialization code here

}
```

Menubehandler für die View – Umschaltung

Mit dem Klassenassistenten fügen wir nun auch Behandler für die Menupunkte ein:



Menubehandler für die Kommandos einfügen

Der Code dafür sieht etwa so aus:

```
void CMainFrame::OnBarview()
{
    m_viewType = Barview;
}

void CMainFrame::OnListview()
{
    m_viewType = Listview;
}
```

Der Code macht natürlich noch nichts nützliches, aber wir können immerhin das Flag umschalten und dieses Flag auch für das Update CommandUI verwenden. Passe die zwei Methoden dementsprechend an:

```
void CMainFrame::OnUpdateListview(CCmdUI* pCmdUI)
{
    int check = 0; // nicht selektiert

    if(Listview == m_viewType)
    {
        check = 1;
    }

    pCmdUI->SetCheck(check);
}

void CMainFrame::OnUpdateBarview(CCmdUI* pCmdUI)
{
    int check = 0; // nicht selektiert

    if(Barview == m_viewType)
    {
        check = 1;
    }

    pCmdUI->SetCheck(check);
}
```

Lass das Programm jetzt laufen. Jetzt kann man immerhin bereits die Toolbar Buttons bedienen. Die Views werden aber selbstverständlich noch nicht erzeugt.

Erzeugen und Umschalten der Views

Eine View wird bereits erzeugt, wir müssen uns aber einen Zeiger darauf merken. Zusätzlich merken wir uns auch einen Zeiger auf eine mögliche Barview (es ist keine Sicht auf eine Bar ;-). Füge also zwei CView* Zeiger als private Datenelemente in die CMainFrame Klasse ein.

```
    CView*   m_barView;
    CView*   m_listView;
```

Natürlich werden beide im Konstruktor initialisiert:

```
CMainFrame::CMainFrame()
           :m_viewType(Listview),
            m_barView(0),
            m_listView(0)
{
}
```

Wenn das MainFrame aktiviert wird, wird die virtuelle Funktion ActivateFrame aufgerufen, die wir mit dem Klassenassistenten überschreiben. Da wir wissen, dass zuerst die Listview aktiv ist, merken wir uns den Zeiger darauf indem wir GetActiveView aufrufen:

```
void CMainFrame::ActivateFrame(int nCmdShow)
{
    CView* test = GetActiveView();
    CFrameWnd::ActivateFrame(nCmdShow);
}
```

Falls wir nun die andere View brauchen, erzeugen wir diese einfach, wobei du vorher noch den Konstruktor von CBarView in den public Teil der Klasse verschieben musst.
Hier ist nun der Code, den ich zum Teil aus der Dokumentation zur MFC habe, also auch nicht einfach so hinschreiben konnte.

```
void CMainFrame::OnBarview()
{
    if(Barview != m_viewType)
    {
        if(0 == m_barView)
        {
            m_barView = new CBarView;
            m_barView->Create(NULL,
                              NULL,
                              AFX_WS_DEFAULT_VIEW,
                              rectDefault,
                              this,
                              AFX_IDW_PANE_FIRST,
                              NULL);
            // das Document holen
            CDocument* doc = GetActiveDocument();
            // und die View in die Liste der
            // Views einfügen
            doc->AddView(m_barView);
        }

        // dieser Zeiger sollte in
        // der Methode ActivateFrame bereits
        // gesetzt worden sein
        ASSERT(0 != m_listView);

        m_barView->SetDlgCtrlID(AFX_IDW_PANE_FIRST);
        m_listView->SetDlgCtrlID(AFX_IDW_PANE_FIRST+1);

        m_listView->ShowWindow(SW_HIDE);
        m_barView->ShowWindow(SW_SHOW);

        SetActiveView(m_barView);
        RecalcLayout();
    }

    m_viewType = Barview;
}
```

```
void CMainFrame::OnListview()
{
    if(Listview != m_viewType)
    {
        // dieser Zeiger sollte in
        // der Methode ActivateFrame bereits
        // gesetzt worden sein
        ASSERT(0 != m_listView);

        m_barView->SetDlgCtrlID(AFX_IDW_PANE_FIRST+1);
        m_listView->SetDlgCtrlID(AFX_IDW_PANE_FIRST);

        m_barView->ShowWindow(SW_HIDE);
        m_listView->ShowWindow(SW_SHOW);

        SetActiveView(m_listView);
        RecalcLayout();
    }

    m_viewType = Listview;
}
```