Hochschule Harz	FB Automatisierung und Informatik
MFC	Grafische Nutzerschnittstellen mit MFC Thema: SDI-Programmierung mit ListView

Versuchsziele

Vertiefung im Verständnis der SDI-Programmierung mit der Tabelle ListView unter Visual Studio 2008.

Überblick:

Diese Übung zeigt die Verwendung der „Tabelle“, ListView in einem SDI-Fenster. Bekannt ist dieses GUI im rechten Teil des Explorers.

Folgende Aufgaben müssen Sie durchführen:

- Erstellen eines SDI-Projektes mit MFC und einem ListViewView (step1)
- Einstellen des Darstellungsmodus der ListView
- Beispielcode in die Tabelle eintragen
- Erweitern:
  - Definieren eine Klasse zur Verwaltung von „Tabellendaten“
  - Definieren der Membervariablen (ArrayList)
  - Laden und Speichern in Serialize vornehmen
  - Im View die Daten lesen und Darstellen
  - Dialogfenster zum Einfügen, Ändern und löschen entwickeln

Grundlagen ListView:

Die Komponente „ListView“ erlaubt die Darstellung von Objekten in einem Fenster. Bekannt durch die rechte Seite im Explorer. Dabei kann man unterschiedliche Darstellungen wählen. Optional kann es eine Dokument-View-Architektur besitzen.

Aufgaben:

1) Grundprojekt erstellen:

Erstellen eines SDI-Projektes mit CListView

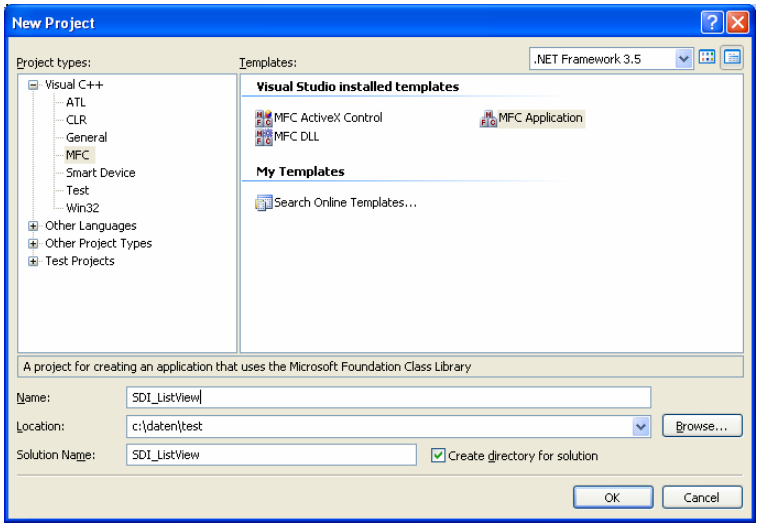


Abbildung 1 MFC-Anwendung erstellen

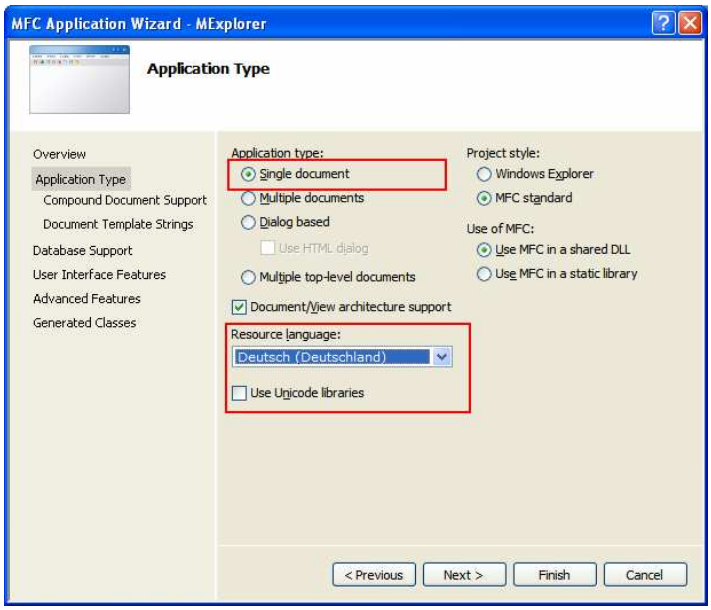


Abbildung 2 SDI-Anwendung

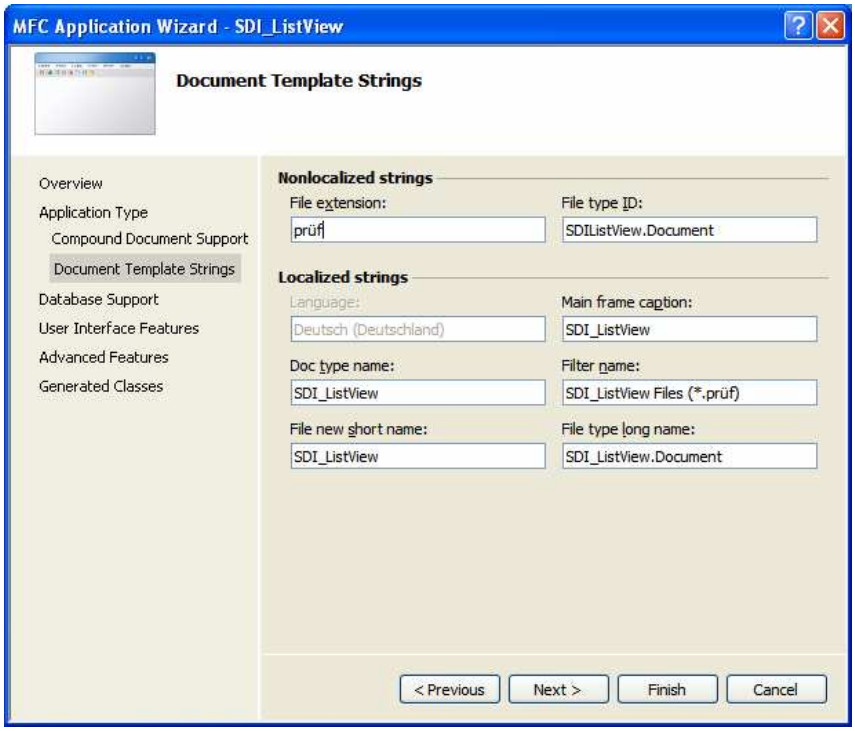


Abbildung 3 Eintragen der Dateierweiterung prüf

Wichtig: Hier Änderung von CView in CListView

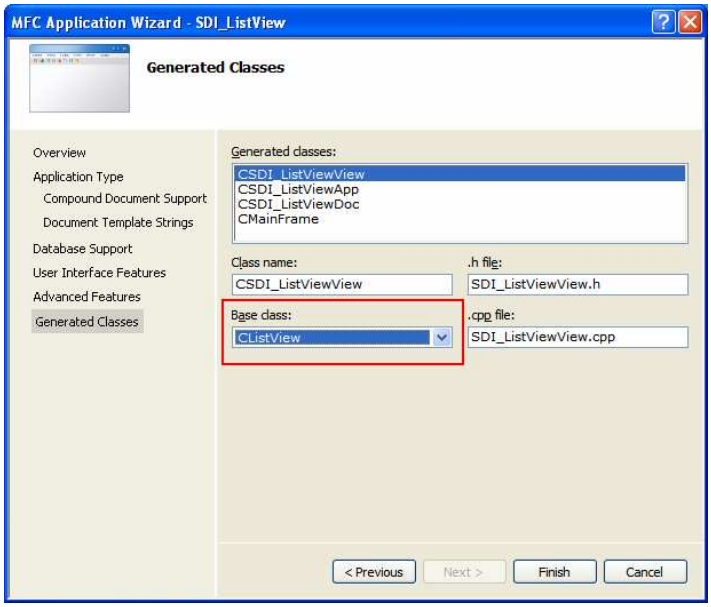


Abbildung 4 Ende des Wizards

Step1

2) ListView-Style erstellen, Datei SDI_ListViewView.cpp

Methode PreCreateWindow

```
cs.style &= ~LVS_TYPEMASK;
cs.style |= LVS_REPORT;
```

3) Spalten setzen, Datei SDI_ListViewView.cpp

Methode OnInitialUpdate

```
   // Spalten für Listenansicht definieren
   GetListCtrl ().InsertColumn (0, "Name", LVCFMT_LEFT, 200);
   GetListCtrl ().InsertColumn (1, "Matrnr", LVCFMT_LEFT, 80);
   GetListCtrl ().InsertColumn (2, "Prüfung", LVCFMT_LEFT, 200);
   GetListCtrl ().InsertColumn (3, "Note", LVCFMT_RIGHT, 100);
```

4) struct in der Datei SDI_ListViewView.h definieren

Die Daten einer Zeile müssen in eine Klasse bzw. einer Struktur gespeichert werden. Die Methode „OnGetdispinfo“ benutzt diese Daten, um für jede Zeile die Tabelle aufzubauen.

```
typedef struct tableITEMINFO {
   CString     sName;
   unsined int Matrnr;
   CString    sPruefung;
   double note;
} tableITEMINFO;
```

5) addItem-Methode einfügen

Die Variable index dient als Schlüssel für die zeile. Man könnte ja auch die tabelle sortieren, dann ist die aktuelle Zeile nicht der Index!

```
BOOL CSDI_ListViewView::addItem(int Index, CString sName, unsigned int Matrnr,
                                CString sPruefung, double note )
{
   tableITEMINFO * pItem;
   // ITEMINFO-Struktur dynamisch anfordern und initialisieren
   try {
      pItem = new tableITEMINFO;
   }
   catch (CMemoryException* e) {
      e->Delete ();
      return FALSE;
   }

   pItem->sName = sName;
   pItem->Matrnr = Matrnr;
   pItem->sPruefung = sPruefung;
   pItem->note = note;


   // Eintrag in Liste-Steuerelement einfügen
   LV_ITEM lvi;
   lvi.mask = LVIF_TEXT | LVIF_IMAGE | LVIF_PARAM;
   //lvi.mask = LVIF_TEXT | LVIF_PARAM;
   lvi.iItem = Index;
   lvi.iSubItem = 0;
   lvi.iImage = 0;
   lvi.pszText = LPSTR_TEXTCALLBACK;
   lvi.lParam = (LPARAM) pItem;

   if (GetListCtrl ().InsertItem (&lvi) == -1) {
      AfxMessageBox("Fehler");
      return FALSE;
   }
```

```
   return TRUE;
} // addItem
```

6) OnGetdispinfo-Methode einfügen

Diese Methode muss an drei Stellen eingefügt werden.

Datei: SDI_ListViewView.h

```
   //{{AFX_MSG(CTableRight)
   afx_msg void OnGetdispinfo(NMHDR* pNMHDR, LRESULT* pResult);
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
```

Datei: SDI_ListViewView.cpp

```
BEGIN_MESSAGE_MAP(CSDI_ListViewView, CListView)
   //{{AFX_MSG_MAP(CTableRight)
   ON_NOTIFY_REFLECT(LVN_GETDISPINFO, OnGetdispinfo)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Und als Quellcode:

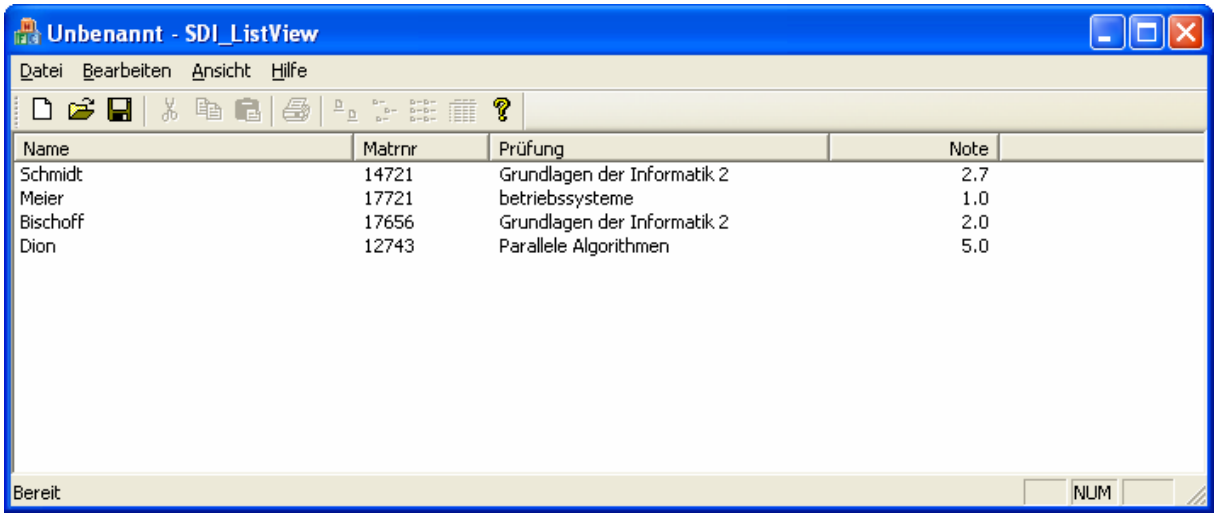
```
void CSDI_ListViewView::OnGetdispinfo(NMHDR* pNMHDR, LRESULT* pResult) {
   LV_DISPINFO* pDispInfo = (LV_DISPINFO*)pNMHDR;
   CString str;
   //AfxMessageBox("GetDispInfo");

   if (pDispInfo->item.mask & LVIF_TEXT) {
      tableITEMINFO* pItem = (tableITEMINFO*) pDispInfo->item.lParam;

      switch (pDispInfo->item.iSubItem) {
      case 0: // Name
         ::lstrcpy (pDispInfo->item.pszText, pItem->sName);
         break;
      case 1: // Matrnr
         str.Format (_T ("%d"), pItem->Matrnr);
         ::lstrcpy (pDispInfo->item.pszText, str);
         break;
      case 2: // sPruefung
         ::lstrcpy (pDispInfo->item.pszText, pItem->sPruefung);
         break;
      case 3: // Note
         str.Format (_T ("%3.1f"), pItem->note);
         ::lstrcpy (pDispInfo->item.pszText, str);
         break;
      }
   }

   *pResult = 0;
}
```

Aufruf des aktuellen Programms:



Nächste mögliche Schritte:

- Liste in der Dokument-Klasse zur Verwaltung und Speicherung
- Sortieren der Spalten

7) Tabelle sortieren

Dazu muss eine Funktion definiert werden, die drei Parameter erhält:

- 1. Objekt
- 2. Objekt
- nach welcher Spalte sortiert wird
- Sortierreihenfolge (optional)

Eintrag in Datei SDI_ListViewView.h:

static int CALLBACK CompareFunc (LPARAM lParam1, LPARAM lParam2,LPARAM lParamSort);

zusätzlich muss noch eine statische Variable definiert werden:

```
static int prevSort;
```

Eintrag in Datei SDI_ListViewView.cpp:

```
BEGIN_MESSAGE_MAP(CSDI_ListViewView, CListView)
   //{{AFX_MSG_MAP(CSDI_ListViewView)
   ON_NOTIFY_REFLECT(LVN_GETDISPINFO, OnGetdispinfo)
   ON_NOTIFY_REFLECT(LVN_COLUMNCLICK, OnColumnClick)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Hier wird nun die Variable initialisiert:

```
int CSDI_ListViewView::prevSort = -1;
```

Hier sind die beiden Methoden:

```
void CSDI_ListViewView ::OnColumnClick(NMHDR* pNMHDR, LRESULT* pResult)
{
   NM_LISTVIEW* pNMListView = (NM_LISTVIEW*) pNMHDR;
```

```
   GetListCtrl ().SortItems (CompareFunc, pNMListView->iSubItem);
   if  (prevSort == pNMListView->iSubItem)
      prevSort = -1;
   else
      prevSort = pNMListView->iSubItem;
   *pResult = 0;
}



// wenn -1  dann ist der erste Wert größer
// wenn +1  dann ist der zweite Wert größer
// wenn 0, dann sind beide Werte gleich
int CALLBACK CSDI_ListViewView::CompareFunc (
LPARAM lParam1, LPARAM lParam2,  LPARAM lParamSort)
{
   tableITEMINFO* pItem1 = (tableITEMINFO*) lParam1;       // werte holen
   tableITEMINFO* pItem2 = (tableITEMINFO*) lParam2;
   int nResult;

   switch (lParamSort) {     // wonach sortieren
   case 0: // name
      nResult = pItem1->sName.CompareNoCase (pItem2->sName);
      //nResult = strcmp(pItem1->sName.GetBuffer(), pItem2->sName.GetBuffer() );
      break;

   case 1: // Matrnr
      nResult = pItem1->Matrnr - pItem2->Matrnr;
      break;

   case 2: // Prüfung
      nResult = pItem1->sPruefung.CompareNoCase (pItem2->sPruefung);
      // nResult = strcmp(pItem1->sPruefung.GetBuffer(), pItem2->sPruefung.GetBuffer() );
      break;

   case 3: // Note
      double n1, n2;
      n1 = pItem1->note;
      n2 = pItem2->note;
      if (n1<n2)
         nResult=1;
      else
         if (n1>n2)
            nResult=-1;
         else
            nResult=0;
      break;
   }
   // war ein Doppelklick schon vorher mit dieser Spalte
   // dann umgekehrte Reihenfolge
   if (prevSort == lParamSort)
      nResult = -nResult;
   return nResult;
}
```