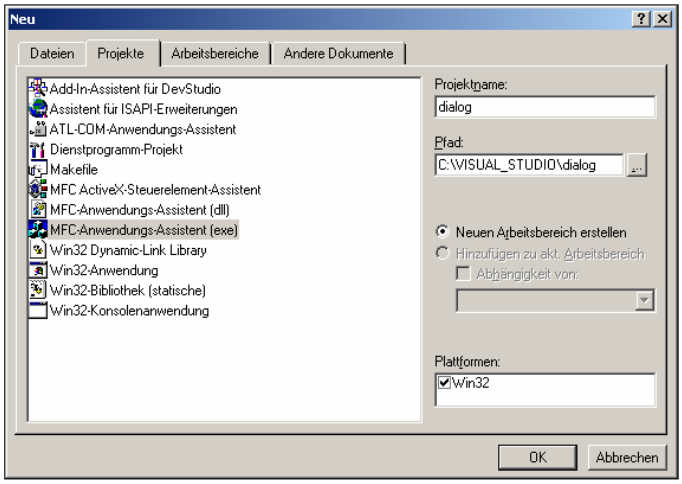
Grundlagen in Visual Studio
MFC, STL

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- mwilhelm@hs-harz.de
- Raum 2.202
- Tel. 03943 / 659 338

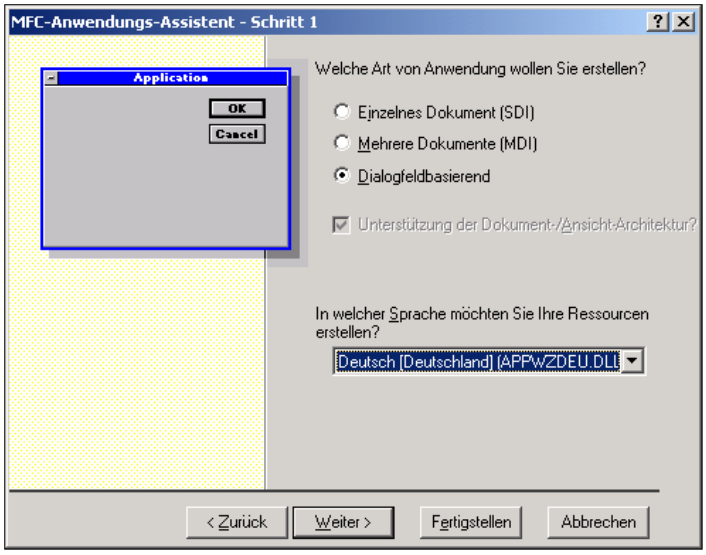
Inhalt

Visual Studio Wizard:

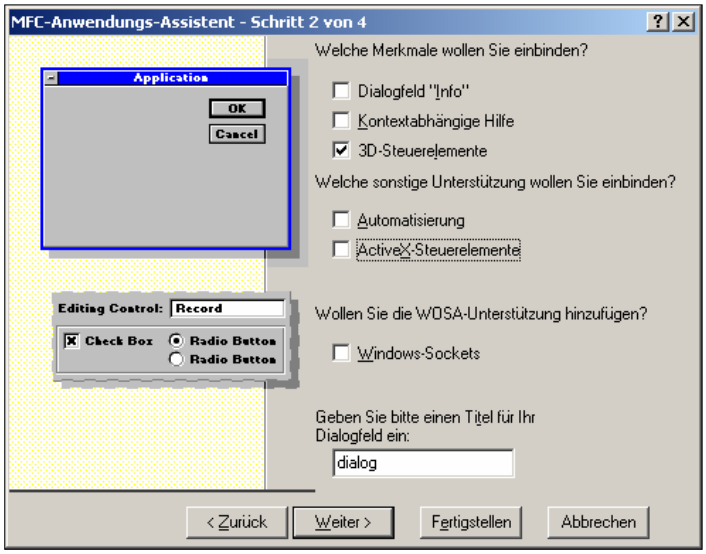
1. Datei / Neu
2. MFC Anwendungs-Assistent (exe)



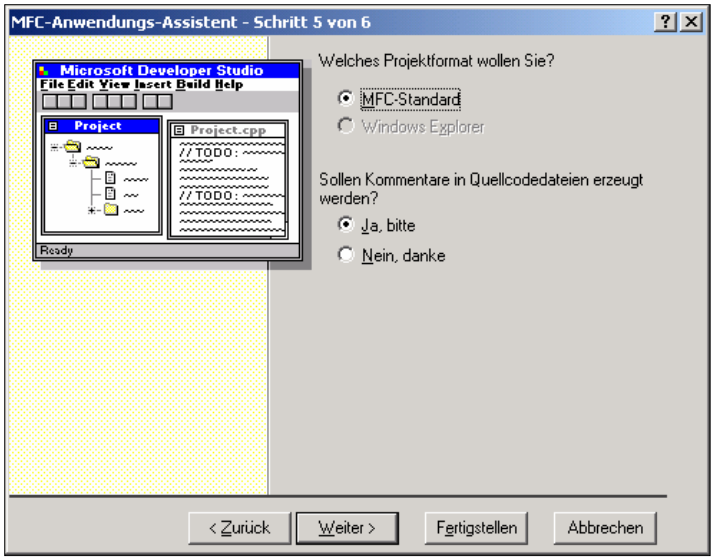
Auswahl: Dialog



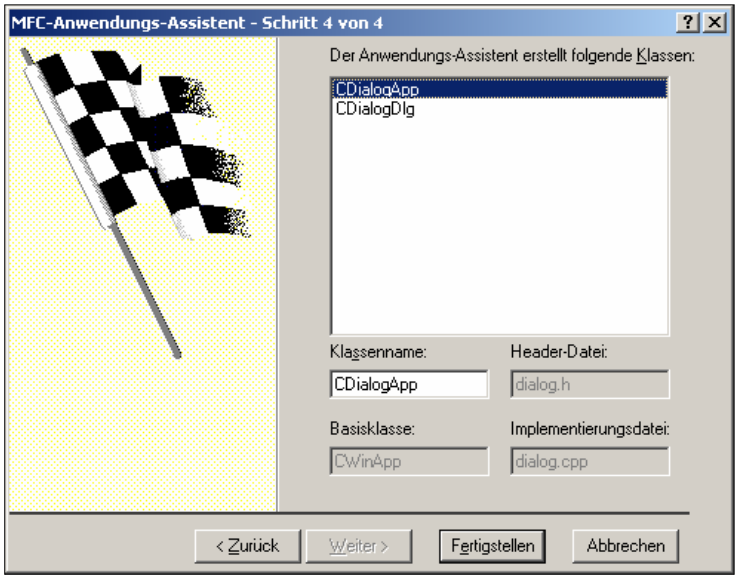
Dialog: Allgemeine Optionen



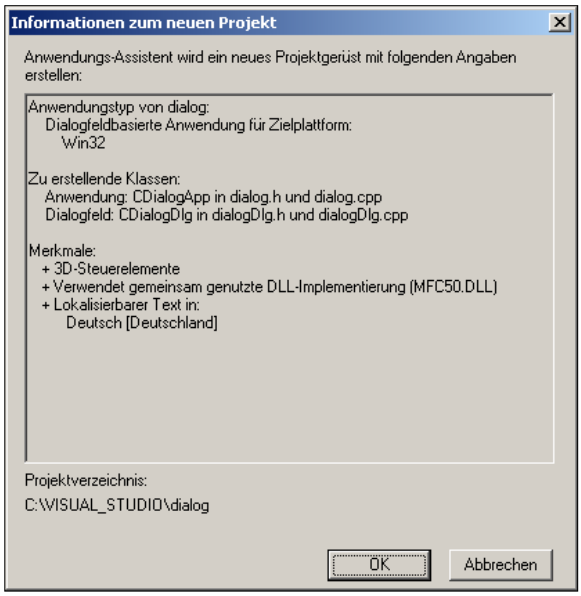
Dialog: Projektformat, Kommentare



Dialog: Klassennamen



Dialog: Ende des Wizards



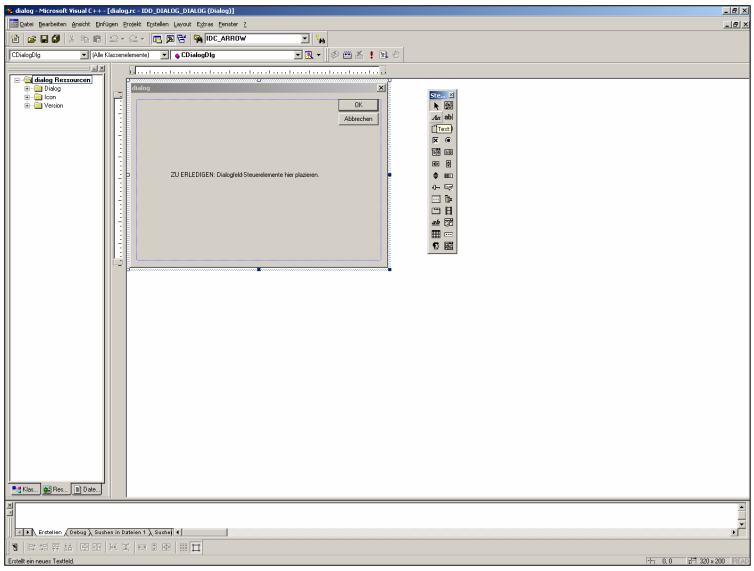
Erzeugte Dateien:

Header-Dateien:
- dialog.h
- dialogDlg.h
- resource.h

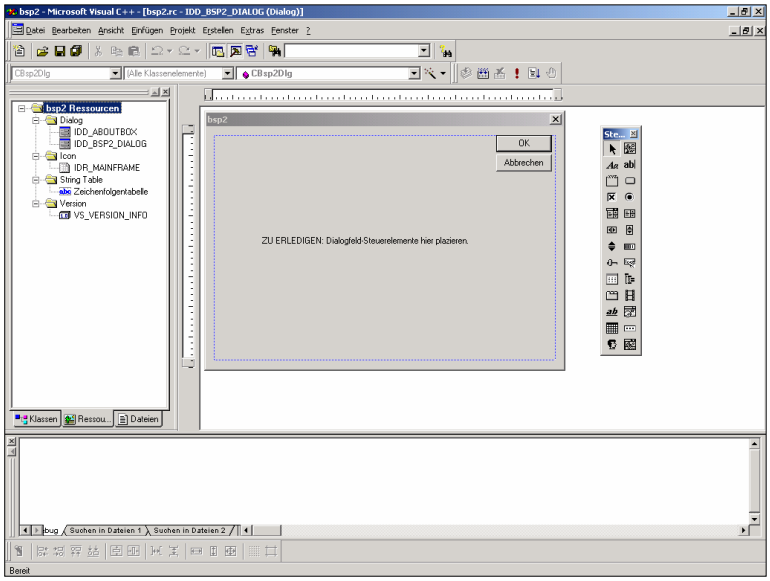
Quellcode-Dateien:
- dialog.cpp — Windowsrahmen, erzeugt das Dialogfenster, ruft es auf
- dialogDlg.cpp — Dialogfenster
- dialog.rc

Klassen:
- CDialogApp — abgeleitet von CWinApp
- CDialogDlg — abgeleitet von CDialog

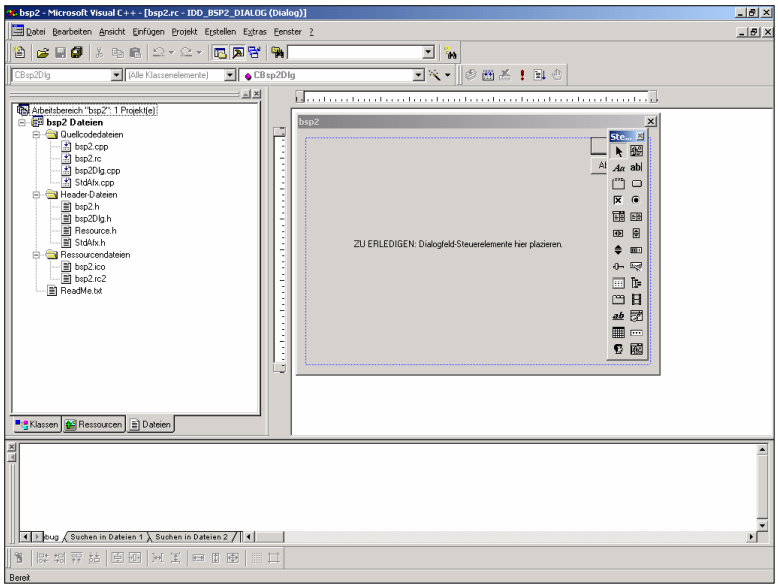
Dialog: IDE, Klassen



Dialog: IDE, Resourcen



Dialog: IDE, Dateien



Wichtige Funktionen

AfxMessageBox ("String");
Zeigt eine modale Meldung

Wichtige Funktionen

int MessageBox(lpText, lpCaption: PChar; uType: UINT);

Flag für Funktionen
MB_ABORTRETRYIGNORE
MB_OK
MB_OKCANCEL
MB_RETRYCANCEL
MB_YESNO
MB_YESNOCANCEL

Flag des Symbols
MB_ICONEXCLAMATION,
MB_ICONWARNING
MB_ICONINFORMATION
MB_ICONASTERISK
MB_ICONQUESTION
MB_ICONSTOP
MB_ICONERROR
MB_ICONHAND

Gültige Rückgabewerte:

IDABORT
IDCANCEL
IDIGNORE
IDNO
IDOK
IDRETRY
IDYES

Flag für den Default-Schalter (Eingabetaste
MB_DEFBUTTON1
MB_DEFBUTTON2
MB_DEFBUTTON3
MB_DEFBUTTON4

Flags der Modalität
MB_APPLMODAL
MB_SYSTEMMODAL
MB_TASKMODAL

Wichtige Funktionen

int MessageBox(lpText, lpCaption: PChar; uType: UINT);

Flag für Funktionen
MB_ABORTRETRYIGNORE
MB_OK
MB_OKCANCEL
MB_RETRYCANCEL
MB_YESNO
MB_YESNOCANCEL
Flag des Symbols
MB_ICONEXCLAMATION,
MB_ICONWARNING
MB_ICONINFORMATION
MB_ICONASTERISK
MB_ICONQUESTION
MB_ICONSTOP
MB_ICONERROR
MB_ICONHAND

Wichtige Funktionen

function MessageBox(hWnd: HWND; lpText, lpCaption: PChar;
uType: UINT): Integer; stdcall;

Flag für den Default-Schalter (Eingabetaste)
MB_DEFBUTTON1
MB_DEFBUTTON2
MB_DEFBUTTON3
MB_DEFBUTTON4

Flags der Modalität
MB_APPLMODAL
MB_SYSTEMMODAL
MB_TASKMODAL

Wichtige Funktionen

function MessageBox(hWnd: HWND; lpText, lpCaption: PChar;
uType: UINT): Integer; stdcall;

Returnwerte
Wenn 0, nicht genügend Speicherplatz

Gültige Rückgabewerte:

IDABORT
IDCANCEL
IDIGNORE
IDNO
IDOK
IDRETRY
IDYES

Specify one of the following flags to indicate the modality of the dialog box:

Flag	Meaning
MB_APPLMODAL	The user must respond to the message box before continuing work in the window identified by the *hWnd* parameter. However, the user can move to the windows of other threads and work in those windows. Depending on the hierarchy of windows in the application, the user may be able to move to other windows within the thread. All child windows of the parent of the message box are automatically disabled, but popup windows are not. MB_APPLMODAL is the default if neither MB_SYSTEMMODAL nor MB_TASKMODAL is specified.

Specify one of the following flags to indicate the modality of the dialog box:

Flag **Meaning**

MB_SYSTEMMODAL Same as MB_APPLMODAL except that the message box has the WS_EX_TOPMOST style. Use system-modal message boxes to notify the user of serious, potentially damaging errors that require immediate attention (for example, running out of memory). This flag has no effect on the user's ability to interact with windows other than those associated with *hWnd*.

MB_TASKMODAL Same as MB_APPLMODAL except that all the top-level windows belonging to the current thread are disabled if the *hWnd* parameter is NULL. Use this flag when the calling application or library does not have a window handle available but still needs to prevent input to other windows in the calling thread without suspending other threads.


```
int retcode = MessageBox("Text","Caption",MB_OK | MB_ICONWARNING);
```

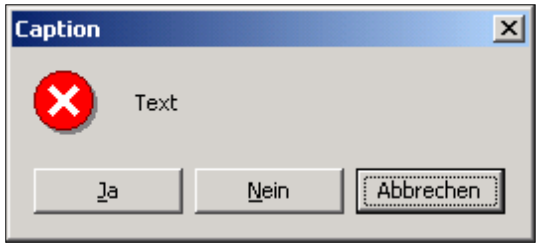

```
retcode = MessageBox("Text","Caption",
                     MB_ABORTRETRYIGNORE | MB_ICONWARNING);
```

```
int retcode = MessageBox("Text","Caption",MB_YESNO | MB_ICONQUESTION);
```


```
switch (retcode) {
    case IDYES:
        AfxMessageBox("YES gedrückt");
        break;
    case IDNO:
        AfxMessageBox("NO gedrückt");
        break;
}
```



```
retcode = MessageBox("Text","Caption",
      MB_YESNOCANCEL | MB_ICONERROR | MB_DEFBUTTON3);
switch (retcode) {
    case IDYES:
        AfxMessageBox("YES gedrückt");
        break;
    case IDNO:
        AfxMessageBox("NO gedrückt");
        break;
    case IDCANCEL:
        AfxMessageBox("CANCEL gedrückt");
        break;
}
```



Dialogelemente in Visual Studio

- Label, CStatic
- Editorfeld, CEdit
- Groupbox
- Schalter, CButton
- Checkbox
- Radiobutton
- ComboBox, CComboBox
- Listenfeld, CListBox
- Bildlaufliste, CScrollBar
- Drehfeld, CSpinButtonCtrl
- Statusanzeige
- Regler
- Zugriffstasten
- Listenelement
- Strukturansicht (Baum)
- Registerkarte
- Animation
- Rich Edit
- Datum- Zeitauswahl
- Kalender
- IP-Adresse
- Erweitertes Kombinationsfeld

Dialogelemente: CStatic

Eigenschaften eines Labels

- ID-Kennung
- Beschriftung
- Textausrichtung
- Enabled
- Kein Umbruch
- Mit Rand
- Vertieft
- Visible

Dialogelemente: CEdit

Eigenschaften des Editorfeldes

- ID-Kennung
- Textausrichtung
- Einzeilig, mehrzeilig
- Kennwort
- Groß- Kleinbuchstaben
- Nummer
- Readonly
- Enabled
- Visible

Dialogelemente: CRadioButton

Eigenschaften eines Radiobuttons

- ID-Kennung
- Groupbox ist nur optisch
- Der Anfang einer Gruppe wird definiert durch Element „Gruppe“ mit dem Wert „TRUE“
- Tabulator-Reihenfolge ändern mit Strg-D

Dialogelemente: CCheckBox

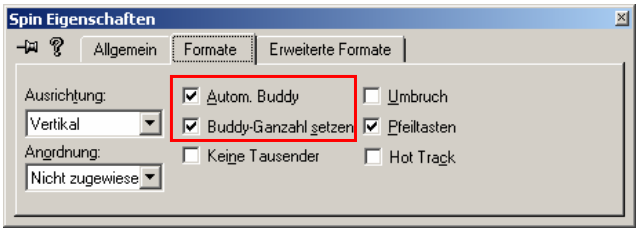
Eigenschaften einer CheckBox

- ID-Kennung
- Groupbox ist nur optisch
- Sichtbar
- Enabled
- Tabulator-Reihenfolge ändern mit Strg-D

Dialogelement: Drehfeld, CSpinButtonCtrl

Eigenschaften eines Drehfeldes

- ID-Kennung
- Verknüpft mit einem Editorfeld
- Änderung per Mausklick
- Buddy-Eigenschaft setzen
- Sichtbar
- Enabled
- Tabulator-Reihenfolge ändern mit Strg-D



Dialogelement: Drehfeld, CSpinButtonCtrl

Einfügen eines Drehfeldes

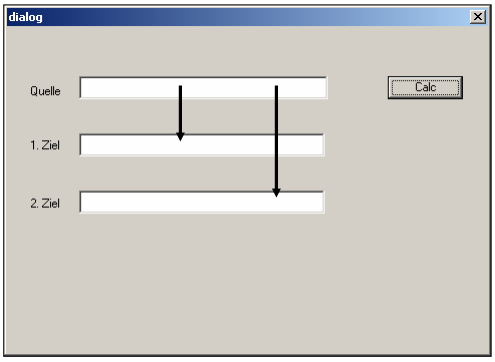
- Einfügen eines Editorfeldes
- Ändern der Number-Eigenschaften im Editorfeld
- Sofort danach Einfügen des Drehfeldes
- Ändern der beiden Buddy-Eigenschaften im Register „Formate“
- Ändern der ID auf z. B. IDC_SPIN_XTICKS
- Damit ist die Verknüpfung zum Editorfeld definiert
- Leider ist die Reihenfolge (Decrement, Increment) falsch
- Abhilfe:
  Dazu muss in der OnInitDialog-Methode folgender Code eingefügt werden:

```
m_SpinXAchse.SetRange(0,12);     // (Membervariable)
m_SpinYAchse.SetRange(0,12);
```


Funktionen mit Dialogelemente

- Member-Variablen
  - Editfeld: CString
  - CheckBox: BOOL
  - ListBox: CString
  - RadioButton: int (Gruppen bezogen, 0..N-1)
- UpdateData(bRichtung)
  - TRUE ⇒ Aus GUI nach Membervariable
  - FALSE ⇒ Aus Membervariable nach GUI
- Ereignisprozedur (Actionlistener)

1. Beispiel: Strings



ID-Kennungen:

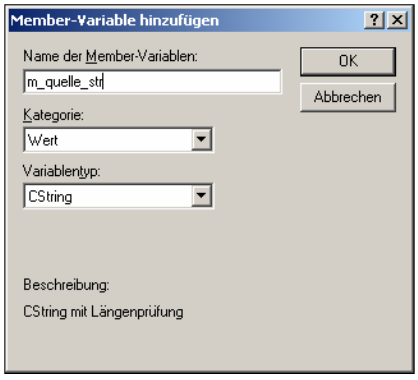
- IDC_QUELLE
- IDC_ZIEL1
- IDC_ZIEL2
- IDC_CALC

Aufgaben

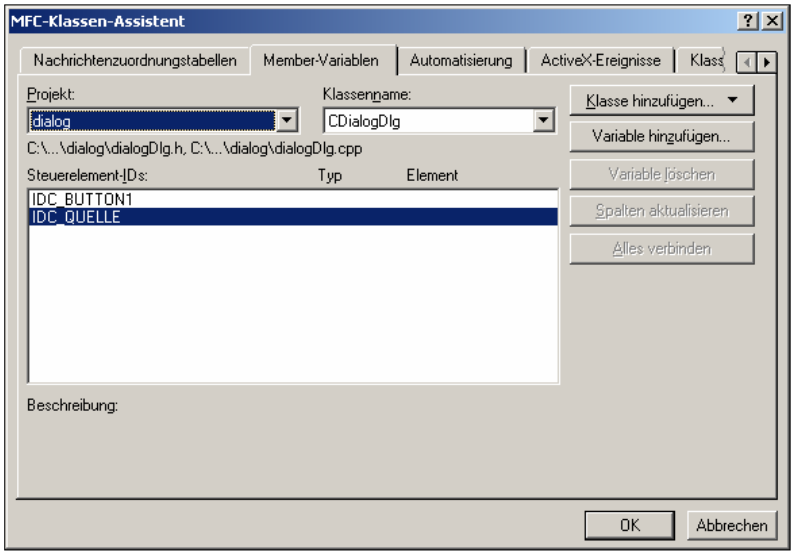
- Transportieren eines String per Schalter
- Transportieren eines String bei Änderung

Erzeugen von Member-Variablen

- Menü Ansicht: Eintrag: Klassen-Assistent (STRG+W)
- Register **Member-Variablen**
- Auswählen des Projektes / Klassennamen
- Variable hinzufügen
- Eintragen des Namens der Variable
- Eventuell Längenbeschränkung

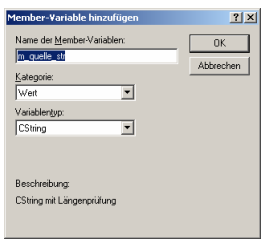


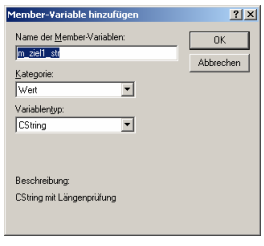
Member-Variablen: m_quelle_str

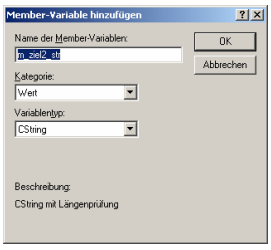


Member-Variablen

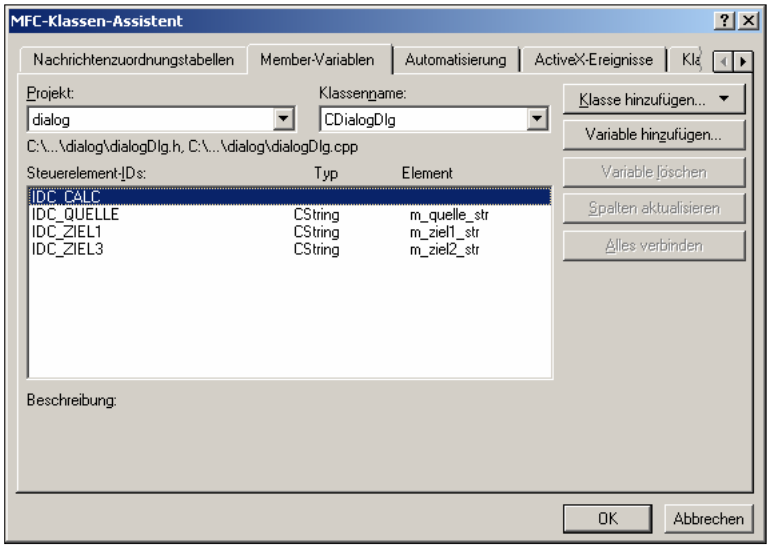
IDC_QUELLE	IDC_ZIEL1	IDC_ZIEL2
m_quelle_str	m_ziel2_str	m_ziel2_str







Member-Variablen: Überblick



Erzeugen von Member-Variablen: Definition

```
void CDialogDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDialogDlg)

        DDX_Text(pDX, IDC_QUELLE, m_quelle_str);
        DDX_Text(pDX, IDC_ZIEL1, m_ziel1_str);
        DDX_Text(pDX, IDC_ZIEL3, m_ziel2_str);

        //}}AFX_DATA_MAP
}
```

Diese Methode bitte nicht direkt aufrufen ! (UpdateData)

Übertragung der Daten

Event-Methode eines RadioButtons

```
int k, iChangeButton;
  // ID holen
k = GetCheckRadioButton( IDC_RADIO1, IDC_RADIO3);
  // welcher Button wurde geändert?
iChangeButton = k - IDC_RADIO1+1;
SetDlgItemInt( IDC_STATIC1, iChangeButton);

// für alle RadioButtons
```


Übertragung der Daten

Event-Methode eines RadioButtons

```
CMyDialog dlg;
dlg.m_radio1 = 2;
dlg.m_radio1 = 0;
if (dlg.DoModal() == IDOK) {
        CString sStr;
        sStr.Format("R1: %d R2: %d",dlg.m_radio1,dlg.m_radio2);
        AfxMessageBox(sStr);
}
```

Initialisierung von GUI-Elementen: 1. Variante

Vor dem Konstruktor wird die Member-Variable definiert

```
BOOL CDialogDlg::OnInitDialog()
{
   m_quelle_str = "Hallo";
   m_ziel1_str = "1. Ziel";
   m_ziel2_str = "2. Ziel";
   CDialog::OnInitDialog();  // // Elemente werden gezeichnet

   return TRUE;
}
```


Initialisierung von GUI-Elementen: 2. Variante

Vor dem Konstruktor wird die Member-Variable definiert

```
BOOL CDialogDlg::OnInitDialog()
{
   CDialog::OnInitDialog();  // Elemente werden gezeichnet
   m_quelle_str = "Hallo";
   m_ziel1_str = "1. Ziel";
   m_ziel2_str = "2. Ziel";
   UpdateData(FALSE);   // // Elemente werden gezeichnet

   return TRUE;
}
```

Übertragung der Daten

Eigenschaften der Liste

```
BOOL CTestDlg::OnInitDialog() {
   CDialog::OnInitDialog();

   CListBox * pListBox = (CListBox *) GetDlgItem(IDC_LIST1);

    pListBox->AddString("a");
    pListBox->AddString("b");
    pListBox->AddString("c");
    pListBox->AddString("d");

   return TRUE;
}
```


Manuelle Übertragung der Daten von einer Liste in ein Label

Eigenschaften der Liste

```
   // OnClick
BOOL CTestDlg::OnSelchangeList1() {

   CListBox * pListBox = (CListBox *) GetDlgItem(IDC_LIST1);
   CString sMsg;

   pListBox->GetText( pListBox->GetCurSel(), sMsg);

   SetDlgItemText( IDC_STATIC1, sMsg);
}
```

Einfache Ausgabe

```
UpdateData(true);
AfxMessageBox(m_liste);
```

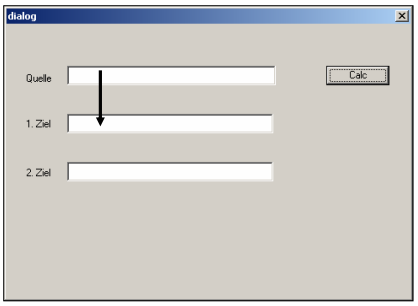
Manuelle Übertragung der Daten von einer Liste in ein Label

Eigenschaften der Liste

```
BOOL CTestDlg::OnSelchangeCombo1() {

    CComboBox * pComboBox;
    pComboBox = (CComboBox *) GetDlgItem(IDC_COMBO1);

    CString sMsg;

     pComboBox->GetLBText( pComboBox->GetCurSel(), sMsg);

     SetDlgItemText( IDC_STATIC1, sMsg);
}
```


1. Beispiel:

Aufgabe: Transportieren eines String per Schalter von IDC_QUELLE nach IDC_ZIEL1 Aktion mittels Schalter



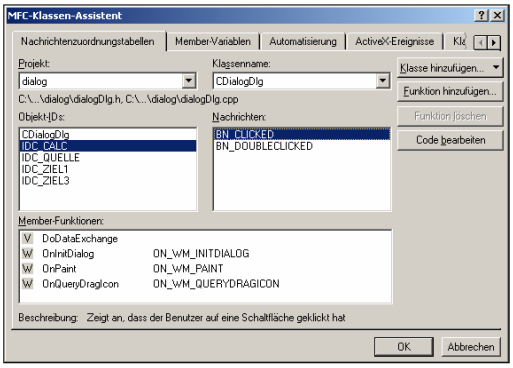
Vorgehensweise:

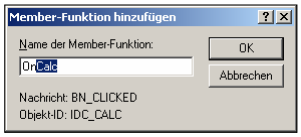
- Membervariablen erzeugen
- Ereignisroutine Schalter definieren
- Datentransfer von einer Variablen zur anderen
- UpdateData aufrufen

Ereignisroutine definieren

Vorgehensweise:

- Aufruf des Dialogfensters mit STRG+W
- Register „Nachrichtenzuordnungstabellen“
- Auswahl Projekt
- Auswahl Klassenname
- Anklicken „IDC_CALC“
- Anklicken „BN_CLICKED“
- Schalter „Funktion hinzufügen“
- Schalter „Code bearbeiten“





Ereignisroutine definieren

1. Version:

```
void CDialogDlg::OnCalc()
{
   m_ziel1_str = m_quelle_str;
}
```

2. Version:

```
void CDialogDlg::OnCalc()
{
   UpdateData(TRUE);
    m_ziel1_str = m_quelle_str;
}
```

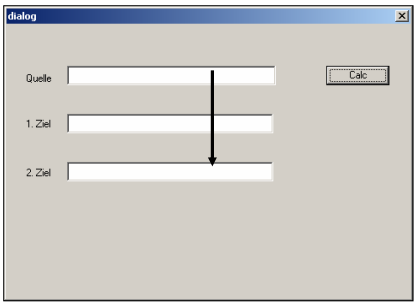
Ereignisroutine definieren

3. Version:

```
void CDialogDlg::OnCalc()
{
      // Übertragung in die MemberVariablen
  UpdateData(TRUE);
   m_ziel1_str = m_quelle_str;
      // Übertragung von den MemberVariablen
   UpdateData(FALSE);

}
```


1. Beispiel:

Aufgabe: Transportieren eines String
von IDC_QUELLE nach IDC_ZIEL2
Aktion bei Änderung



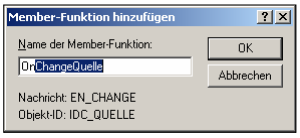
Vorgehensweise:

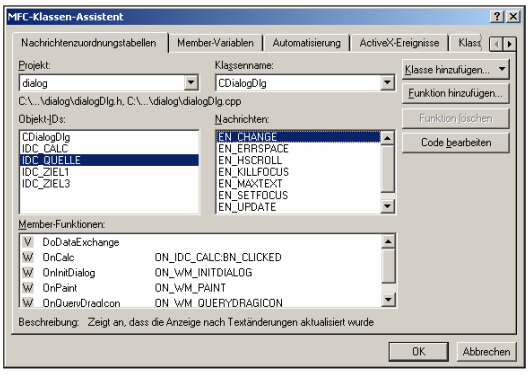
- Membervariablen erzeugen
- Ereignisroutine OnChange definieren
- Datentransfer von einer Variablen zur anderen
- UpdateData aufrufen

Ereignisroutine definieren

Vorgehensweise:

- Aufruf des Dialogfensters mit STRG+W
- Register „Nachrichtenzuordnungstabellen“
- Auswahl Projekt
- Auswahl Klassenname
- Anklicken „IDC_CALC“
- Anklicken „EN_CHANGE“
- Schalter „Funktion hinzufügen“
- Schalter „Code bearbeiten“





Ereignisroutine definieren

```
void CDialogDlg::OnChangeQuelle()
{
      // Übertragung in die MemberVariablen
   UpdateData(TRUE);

   m_ziel2_str = m_quelle_str;

      // Übertragung von den MemberVariablen
   UpdateData(FALSE);
}
```

Übungen

- Dialogfenster mit Übertragung der Strings
- Dialogfenster mit Übertragung einer Zahl
  - IDC_QUELLE eintragen einer Zahl
  - Wenn numerisch, dann Ausgabe: „Ergebnis: 123“
- Dialogfenster mit Übertragung von berechneten Zahlen
  - Zwei Editfelder (Input, A und B)
  - Eine ComboBox (mathematische Operation)
  - Ein Schalter
  - Ein Editfeld (Output C) in Abhängigkeit der Operation
- .

Zusatzfunktionen

sscanf, swscanf
Read formatted data from a string.

int sscanf(const char *buffer, const char *format [, argument] ...);

int swscanf(const wchar_t *buffer, const wchar_t *format [, argument] ...);

Headerdateien:
- sscanf <stdio.h> ANSI, Win 95, Win NT
- swscanf <stdio.h> or <wchar.h> ANSI, Win 95, Win NT

Zusatzfunktionen

Parameters:

buffer: Stored data
format: Format-control string
argument: Optional arguments

The **sscanf** function reads data from buffer into the location given by each argument. Every argument must be a pointer to a variable with a type that corresponds to a type specifier in format. The format argument controls the interpretation of the input fields and has the same form and function as the format argument for the scanf function; see scanf for a complete description of format. If copying takes place between strings that overlap, the behavior is undefined.

swscanf is a wide-character version of sscanf; the arguments to swscanf are wide-character strings. sscanf does not handle multibyte hexadecimal characters. swscanf does not handle Unicode fullwidth hexadecimal or “compatibility zone„ characters. Otherwise, swscanf and sscanf behave identically.

Zusatzfunktionen

Bemerkungen:

The sprintf function formats and stores a series of characters and values in buffer. Each argument (if any) is converted and output according to the corresponding format specification in format. The format consists of ordinary characters and has the same form and function as the format argument for printf. A null character is appended after the last character written. If copying occurs between strings that overlap, the behavior is undefined.

swprintf is a wide-character version of sprintf; the pointer arguments to swprintf are wide-character strings. Detection of encoding errors in swprintf may differ from that in sprintf. swprintf and fwprintf behave identically except that swprintf writes output to a string rather than to a destination of type FILE.

Beispiele:

```
#include <stdio.h>
void main( void )
{
  char  tokenstring[] = "15 12 14...";
  char  s[81];
  char  c;
  int   i;
  float fp;

  sscanf( tokenstring, "%s", s );
  sscanf( tokenstring, "%c", &c );
  sscanf( tokenstring, "%d", &i );
  sscanf( tokenstring, "%f", &fp );

  printf( "String    = %s\n", s );
  printf( "Character = %c\n", c );
  printf( "Integer:  = %d\n", i );
  printf( "Real:     = %f\n", fp );
}
```

Ausgabe

```
String    = 15
Character = 1
Integer:  = 15
Real:     = 15.000000
```

See Also
- sscanf
- fscanf,
- scanf,
- sprintf,
- _snprintf

Zusatzfunktionen

sprintf, swprintf

Write formatted data to a string.

```
int sprintf( char *buffer, const char *format [, argument] ... );
int swprintf( wchar_t *buffer, const wchar_t *format [, argument] ... );
```

Header-Dateien

sprintf	<stdio.h>	ANSI, Win 95, Win NT
swprintf	<stdio.h> or <wchar.h>	ANSI, Win 95, Win NT

Zusatzfunktionen

Return Value

sprintf returns the number of bytes stored in buffer, not counting the terminating null character. swprintf returns the number of wide characters stored in buffer, not counting the terminating null wide character.

Parameter:

- buffer: Storage location for output
- format: Format-control string
- argument: Optional arguments

Beispiel:

```
#include <stdio.h>
void main( void )
{
  char  buffer1[50];
  char  buffer2[50];
  char  buffer3[50];
  char  s[] = "computer“;
  int   i = 35;
  float fp = 1.7320534f;

  sprintf( buffer1,     "String:    %s", s );
  sprintf( buffer2,     “Integer:    %d", i );
  sprintf( buffer3,     “Real:    %f", fp );

  AfxMessageBox(buffer1);  // etc
}
```

Beispiel:

```
#include <stdio.h>
void main( void )
{
  char  buffer[200],
  char  s[] = "computer“
  char c = 'l';
  int   i = 35, j;
  float fp = 1.7320534f;

  j  = sprintf( buffer,     "\tString:    %s\n", s );
  j += sprintf( buffer + j, "\tCharacter: %c\n", c );
  j += sprintf( buffer + j, "\tInteger:   %d\n", i );
  j += sprintf( buffer + j, "\tReal:      %f\n", fp );

  printf( "Output:\n%s\ncharacter count = %d\n", buffer, j );
}
```


Ausgabe:

```
Output:
  String:    computer
  Character: l
  Integer:   35
  Real:      1.732053

character count = 71
```

Format float

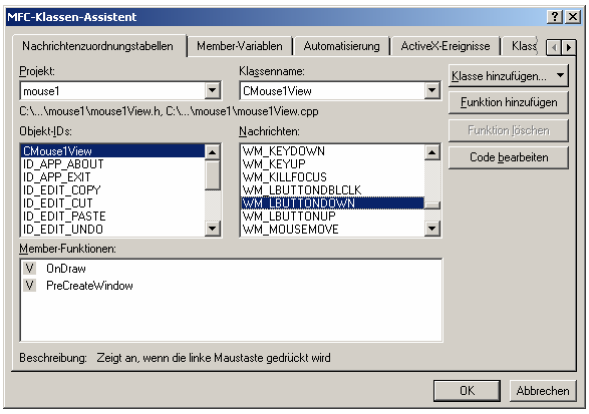
```
float x=1234.123456789;
char  buffer[200];
sprintf( buffer, "x:   %14.8f\n", x);
AfxMessageBox(buffer);
```

Format double

```
double x=1234.123456789;
char  buffer[200];
sprintf( buffer, "x:   %14.8lf\n", x);
AfxMessageBox(buffer);
```



```
void CDialogDlg::OnCalc()
{
  int   iNumber;
  float fNumber;
  char  buffer[200];
  int j;

   UpdateData(TRUE);

    k = sscanf(m_str_name1, "%d", &iNumber );
    AfxMessageBox(m_str_name1);
    if (k==1) {
       j = sprintf( buffer, "Integer:   %d\n", iNumber );
       AfxMessageBox(buffer);
    }
}
```




```
void CMouse1View::OnLButtonDown(UINT nFlags, CPoint
point)
{
        // TODO: Code für die Behandlungsroutine für
Nachrichten hier einfügen und/oder Standard aufrufen

        CView::OnLButtonDown(nFlags, point);
}
```

```
CString s;
int i;
i=33;
s.Format("Ausgabe i:: %d", i );

        AfxMessageBox(s);
```