Grundlagen in Visual Studio MFC und .net

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

Lernziele

- Grundlegende Kenntnisse im Aufbau von Windows-programmierung
- Aufbau der Microsoft Foundation Class
  - Fenster-Klassen
  - Dialog, SDI
  - Allgemeine Klassen (CString, CFile)
  - Allgemeine Dialogfenster (AFXMessageBox etc.)

Inhalt: MFC

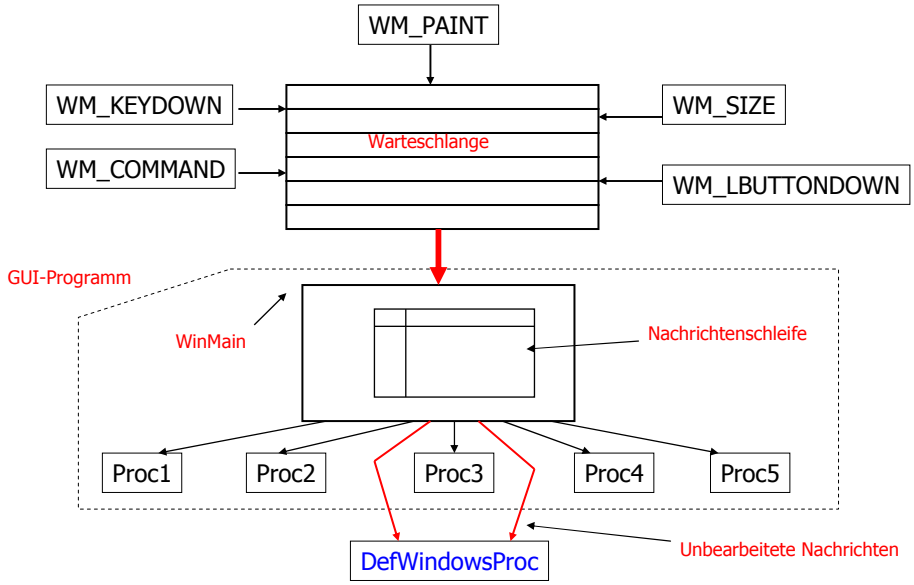
Literatur

- **Visual C++ 6**, Dr. Susanne Vogel, ISBN 3-8287-5019-2
- **Visual C++ 6**, Richard C. Leinecker, Tom Archer, mitp-Verlag, ISBN 3-8266-0902-6, 1. Auflage 2002
- Windows-Programmierung mit MFC, Jeff Prosise, 2. Auflage, ISBN 3-86063-434-8
- **Visual C++ 6**, Steven Holzner, Sybex-Verlag, 1. Auflage 1998, ISBN 3-8155-7009-3
- **Visual C++ 6 in 21 Tagen**,
- **Microsoft Foundation Class in 21 Tagen**,

GUI-Programmierung

- Standardprogramme haben einen sequentiellen Ablauf
- Der Ablauf von Standardprogramme wird durch Parameter gesteuert.
- GUI-Programmierung à la Java, Visual Studio, Qt, Delphi, .net sind dagegen „Ereignisgesteuert“. Der Anwender oder das Betriebsystem steuern den Verlauf des Programms.
- Es ist unerheblich für die Programmiersprache, ob ein prozeduraler oder objektorientierter Ansatz gewählt wird.
- Beispiele für die Ereignisse ?

GUI-Programmierung



WINUSER.h

WM_CREATE
WM_DESTROY
WM_MOVE
WM_SIZE
WM_ACTIVATE

WM_SETFOCUS
WM_ENABLE
WM_SETTEXT
WM_GETTEXT
WM_GETTEXTLENGTH
WM_PAINT
WM_CLOSE
WM_QUIT

WM_SETCURSOR
WM_CHILDACTIVATE

WM_KEYFIRST
WM_KEYDOWN
WM_KEYUP
WM_CHAR

WM_INITDIALOG
WM_HSCROLL
WM_VSCROLL
WM_INITMENU
WM_INITMENUPOPUP
WM_MENUDRAG

WM_MOUSEFIRST
WM_MOUSEMOVE
WM_LBUTTONDOWN
WM_LBUTTONUP
WM_LBUTTONDBLCLK
WM_RBUTTONDOWN
WM_RBUTTONUP
WM_RBUTTONDBLCLK
WM_MBUTTONDOWN
WM_MBUTTONUP
WM_MBUTTONDBLCLK

WM_MDICREATE
WM_MDIDESTROY
WM_MDIACTIVATE
WM_MDIRESTORE
WM_MDINEXT
WM_MDIMAXIMIZE
WM_MDITILE
WM_MDICASCADE
WM_MDIICONARRANGE
WM_MDIGETACTIVE

WM_CUT
WM_COPY
WM_PASTE
WM_CLEAR
WM_UNDO
WM_PRINT

WINUSER.h

Besondere Konstante:

```
#if (_WIN32_WINNT >= 0x0400) || (_WIN32_WINDOWS > 0x0400)
  #define WM_MOUSEWHEEL             0x020A
  #define WM_MOUSELAST              0x020A
#else
  #define WM_MOUSELAST              0x0209
#endif /* if (_WIN32_WINNT < 0x0400) */


#define WM_USER                     0x0400
```

Wichtige Headerdateien

- Windows.h
- Winbase.h
- Windef.h
- Wingdi.h
- Winresrc.h
- Winuser.h
- shellapi.h

Windows ohne MFC 1

```
#include "stdafx.h"

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR lpCmdLine,
                     int  nCmdShow )
{
  WNDCLASS wc;
  HWND hwnd;
  MSG msg;

  wc.style = 0;
  wc.lpfnWndProc = (WNDPROC) WndProc;
  wc.cbClsExtra = 0;
  wc.cbWndExtra = 0;
  wc.hInstance = hInstance;
  wc.hIcon = LoadIcon(NULL, IDI_WINLOGO);
  wc.hCursor = LoadCursor(NULL,IDC_ARROW);
  wc.hbrBackground = (HBRUSH) (COLOR_WINDOW + 1);
  wc.lpszMenuName = NULL;
  wc.lpszClassName = "MeineFensterklasse";
  RegisterClass(&wc);

  hwnd = CreateWindow(
                "MeineFensterklasse",
                "Windows ohne MFC",
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                CW_USEDEFAULT,
                CW_USEDEFAULT,
                CW_USEDEFAULT,
                HWND_DESKTOP,
                NULL,
                hInstance,
                NULL);

  ShowWindow(hwnd, nCmdShow);
  UpdateWindow(hwnd);

  while (GetMessage(&msg, NULL, 0, 0)) {
            TranslateMessage( &msg );
            DispatchMessage( &msg);
  }
  return msg.wParam;

}
```

Windows ohne MFC 2

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT message, WPARAM wparam, LPARAM lparam) {
 PAINTSTRUCT ps;
 HDC hdc;

 switch (message) {
  case WM_PAINT:
        hdc = BeginPaint(hwnd, &ps);
        Ellipse(hdc, 0,0, 300,400);
       EndPaint(hwnd, &ps);
       UpdateWindow(hwnd);
                        return 0;

  case WM_DESTROY:
         PostQuitMessage(0);
         return 0;
 }  // switch
 return DefWindowProc(hwnd, message, wparam, lparam);
};
```


Windows ohne MFC 3

```
// Prototypen in WinUser.h

WINUSERAPI  BOOL  WINAPI  GetMessageA(
            LPMSG lpMsg,
            HWND hWnd ,
            UINT wMsgFilterMin,
            UINT wMsgFilterMax);


typedef LRESULT
                    (CALLBACK* WNDPROC)
                    (HWND, UINT, WPARAM, LPARAM);
```

Windows mit MFC 1: Hello.h

```
// programmspezifische Version der Anwendungsklasse
class CMyApp : public CWinApp
{
  public:
    virtual BOOL InitInstance ();
};



// programmspezifische Version der Klasse für das Hauptfenster
class CMainWindow : public CFrameWnd
{
 public:
    CMainWindow ();          // Konstruktor
 protected:
    afx_msg void OnPaint ();  // Behandlungsroutine für WM_PAINT
    DECLARE_MESSAGE_MAP ()
};
```


Windows mit MFC 2: Hello.cpp

```
#include <afxwin.h>
#include "Hello.h"

CMyApp myApp;

BOOL CMyApp::InitInstance ()
{
   m_pMainWnd = new CMainWindow;
   m_pMainWnd->ShowWindow (m_nCmdShow);
   m_pMainWnd->UpdateWindow ();
   return TRUE;
}
        //   Makro
BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
   ON_WM_PAINT ()
END_MESSAGE_MAP ()

CMainWindow::CMainWindow ()
{
   Create (NULL,  "Das erste Programm“ );   // weitere Konstruktoren existieren
}
```

Windows mit MFC 3 : Hello.cpp

```
void CMainWindow::OnPaint ()
{
   CPaintDC dc (this);                  // Definition und Erzeugen eine Paint-Objects

   CRect rect;                          // Rechteck Bereich
   GetClientRect (&rect);               // Hole den aktuellen Bereich

   dc.DrawText (
            "Hello, MFC",               // Text
            -1,                         // Anzahl der Zeichen
            &rect,                      // Zeichenbereich
            DT_SINGLELINE | DT_CENTER | DT_VCENTER
    );
}
```


Konzepte der MFC

- MFC ist eine objektorientierte Klassenbibliothek
- Sie baut eine Hülle um die API-Funktionen
- Sie enthält ca. 200 Klassen (CPoint, CWnd)
- MFC bietet ein Programmgerüst
- MFC bietet allgemeine Klassen (CString, CList, CArray)
- MFC bietet wiederverwendbare Klassen
- MFC stellt alle Dialogelemente über Klassen zur Verfügung
- MFC ohne großen Überbau:
  - ohne zusätzliche Belastung des Prozessors
  - ohne übermäßig zusätzlichen Speicherplatz
- Fensterparameter werden im Betriebssystem gehalten
- Fast alle MFC-Attribute sind public, keine get-set-Methoden
- Dokument / View Architektur
- Druckvorschau

Aufbau der MFC (1)

- Klasse CMyApp abgeleitet von CWinApp (genau einmal)
- CMyApp definiert den Rahmen
- Klasse CMainWindow abgeleitet von CFrameWndApp
- Es gibt genau ein globales Objekt von CMyApp
- CWinApp definiert die Nachrichtenschleife
- CWinApp hat virtuelle Methoden
  - InitInstance
  - ExitInstance (CWinApp::ExitInstance(....)
  - OnIdle
  - Run
  - PreTranslateMessage
- CMainWindow steuert die Fensterdarstellung (kann mehrfach vorkommen)
- InitInstance erzeugt ein Fenster und setzt Visible-Bit und kopiert seine Instance in die Variable m_pMainWnd (Member, pointer, Main, Window)
- Es gibt keine sichtbare WinMain oder main-Methode !!!
- WinMain steht in AfxWinMain

Aufbau der MFC (2)

- Aufgaben von AfxWinMain
  - ruft AfxWinInit auf
  - initialisiert das Programmgerüst
  - erhält hInstance, trägt den Wert in die Datenfelder
  - Aufruf der Methode InitApplication (Win16)
  - Aufruf der Methode InitInstance
  - Starten des ersten Thread pThread->Run();
- Aufgaben von CWnd
  - Stellt Variablen und Methoden für die Fensterwaltung zur Verfügung
  - Ableitung: Klasse CFrameWnd
  - Muss im Konstruktor Create oder CreateEx aufrufen
  - Jede Anwendung leitet eine Instanz ab (CMainWindow)
  - Durch Konstruktor wird das Fenster definiert (Abmessungen, Styles, Icons etc.).

Hello World

- „Hello World“ aus der Konsole

```
#include „stdafx.h“

int main( int argc, char *argv[]) {
    char line[20];
    printf("Hallo Welt!\n");

    gets( line );
}
```

Eigenschaften:

- Echte 32-Bit Anwendung
- Keine GUI
- aber Unterprogramme

ASSERT (1)

```
#include "stdafx.h"

#include "Math.h"

double getsqrt(double x) {
    return sqrt(x);
}
```

```
int main(int argc, char* argv[])
{
    double x,y;

    x = 33;
    y = getsqrt(x);
    printf("x: %f    y: %f\n",x,y);

    x = -33;
    y = getsqrt(x);
    printf("x: %f    y: %f\n",x,y);
    return 0;
}
```

ASSERT (2)

```
#include "stdafx.h"
#include "assert.h"

#include "Math.h"

double getsqrt(double x) {
   assert(x>=0.0);
   return sqrt(x);
}
```

```
int main(int argc, char* argv[])
{
    double x,y;

    x = 33;
    y = getsqrt(x);
    printf("x: %f    y: %f\n",x,y);

    x = -33;
    y = getsqrt(x);
    printf("x: %f    y: %f\n",x,y);
    return 0;
}
```

- Assert berechnet einen Ausdruck. Wenn das Ergebnis 0 oder false ist, wird eine Debug-Fehlermeldung ausgegeben
- Nicht in der Release-Version

ASSERT (3)

```
#include "stdafx.h"
#include "Math.h"

double getsqrt(double x) {
   if ( x<0.0 ) {
      Message(.....)
   }
   return sqrt(x);
}
```

```
#include "stdafx.h"

#include "Math.h"

double getsqrt(double x) {
   #ifdef CHECK
       if ( x<0.0 ) {
       Message(.....)
       }
   #endif
   return sqrt(x);
}
```

- Assert berechnet einen Ausdruck. Wenn das Ergebis 0 oder false ist, wird eine Debug-Fehlermeldung ausgegeben
- Mit #ifdef auch in der Release-Version

ASSERT (4)

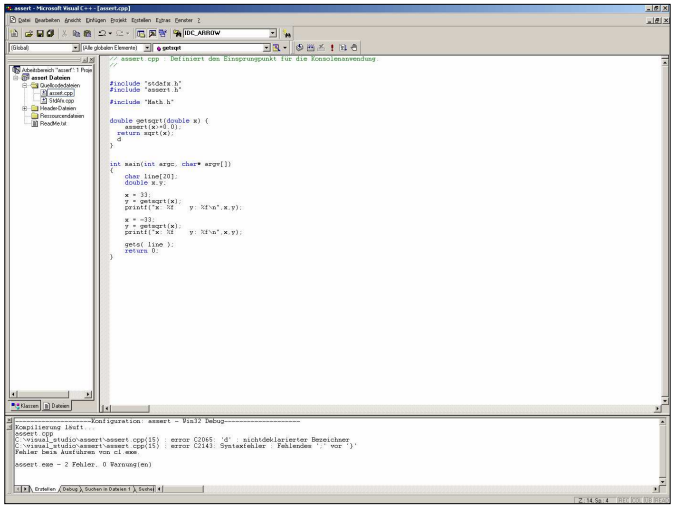
Typen des Assert-Makros:

- Assert(Ausdruck), "assert.h“
- _ASSERT, crtdbg.h,
  - LIBCD.LIB, Single thread static library, debug version
  - LIBCMTD.LIB, Multithread static library, debug version
  - MSVCRTD.LIB, Import library for MSVCRTD.DLL, debug version
- ASSERT, MFC

CAge* pcage = new CAge(21);
CObject. ASSERT(pcage != NULL)

IDE mit Quellcode

- Verwaltung der Klassen
- Verwaltung der Resourcen
- Anzeige des Quellcodes
- Ergebnisfenster beim Übersetzen



IDE von Visual Studio

Hotkeys

- F5 Starten des Programms
- **CTRL+F5 Starten des Programms**
- F9 Setzen eines Breakpoints
- F10 Methode, Aufruf überspringen
- F11 in Aufruf springen
- CTRL+F10 Ausführen bis Cursor
- CTRL+F7 Modul kompilieren
- F7 Modul erstellen
- Alt+F7 Einstellungen (Header, Lib)
- Alt+F9 Liste aller Haltepunkte