GUI oder was?

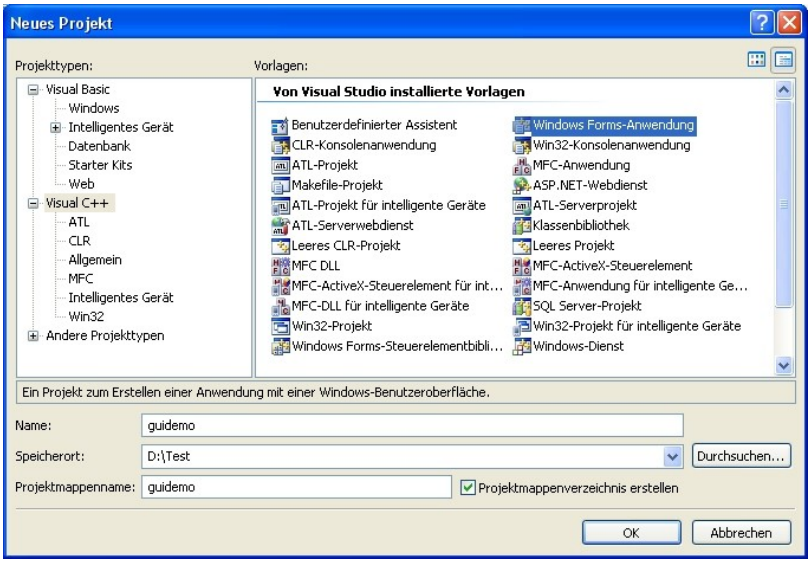
Die Programmierung mit Fenstern ist sicherlich reizvoller, als die Programmierung für die Befehlszeile. In der Vergangenheit war die GUI-Programmierung unter Windows jedoch noch sehr komplex -mit einer Ausnahme: Die Programmierung in Visual Basic (kurz: VB).

Durch die neuen Generationen des Visual Studio (kurz: VS) hat Microsoft jedoch auch für die GUI-Programmierung in C++ einen großen Schritt zu einer vereinfachten Programmerstellung hin getan. Die Gestaltung von Bedienoberflächen und die Programmierung ist nun ähnlich einfach wie in VB.

Im Folgenden möchte ich die Erstellung eines GUI-Programmes als Windows Form Projekt (WFP) vorstellen und anchließend noch einige Worte zur „Philosophie“ der Fenster-Programmierung verlieren.

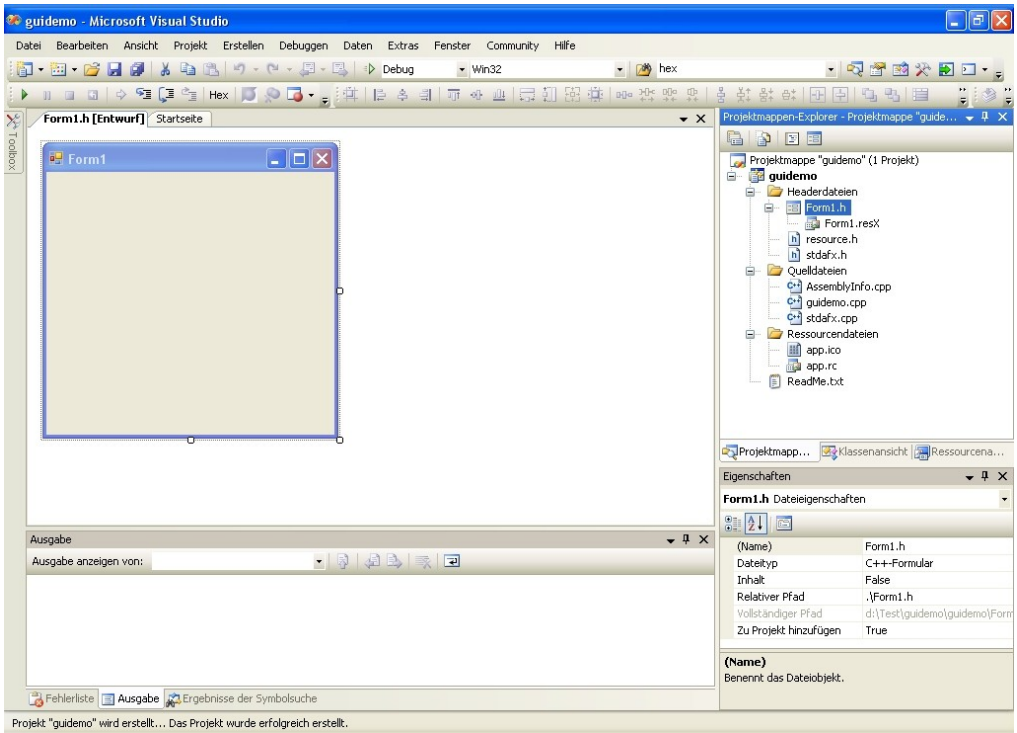
Ein Windows Form Projektes

In der Entwicklungsumgebung wird ein neues Projekt erstellt. Hierbei wird die Vorlage „Windows Forms Anwendung“ aus dem Bereich „Visual C++“ ausgewählt.



Nachdem man dies mit dem „OK-Button“ bestätigt hat, präsentiert sich die IDE wie im nächsten Bild.

Es wurde ein leeres Fenster mit dem Namen **`Form1`** erzeugt und im Projektexplorer tauchen zahlreiche neue Dateien auf, von denen aber nur wenige jetzt von Bedeutung sind.



Wichtig ist die Datei **`Form1.h`**. In dieser Datei befindet sich der Programmcode für das betreffende Fenster. Das Gerüst würde automatisch von Visual Studio erzeugt und ist auf der nächsten Seite dargestellt und erläutert.

```
#pragma once

namespace guidemo {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Zusammenfassung für Form1
    ///
    /// Warnung: Wenn Sie den Namen dieser Klasse ändern, müssen Sie auch die Ressourcendateiname
    ///          -Eigenschaft für das Tool zur Kompilierung verwalteter Ressourcen ändern,
    ///          das allen RESX-Dateien zugewiesen ist, von denen diese Klasse abhängt.
    ///          Anderenfalls können die Designer nicht korrekt mit den lokalisierten Ressourcen
    ///          arbeiten, die diesem Formular zugewiesen sind.
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    public:
          Form1(void)
          {
                InitializeComponent();
                //
                //TODO: Konstruktorcode hier hinzufügen.
                //
          }

    protected:
          /// <summary>
          /// Verwendete Ressourcen bereinigen.
          /// </summary>
          ~Form1()
          {
                if (components)
                {
                      delete components;
                }
          }

    private:
          /// <summary>
          /// Erforderliche Designervariable.
          /// </summary>
          System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
          /// <summary>
          /// Erforderliche Methode für die Designerunterstützung.
          /// Der Inhalt der Methode darf nicht mit dem Code-Editor geändert werden.
          /// </summary>
          void InitializeComponent(void)
          {
                this->SuspendLayout();
                //
                // Form1
                //
                this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                this->ClientSize = System::Drawing::Size(292, 266);
                this->Controls->Add(this->button1);
                this->Name = L"Form1";
                this->Text = L"Form1";
                this->ResumeLayout(false);

          }
#pragma endregion
    };
}
```

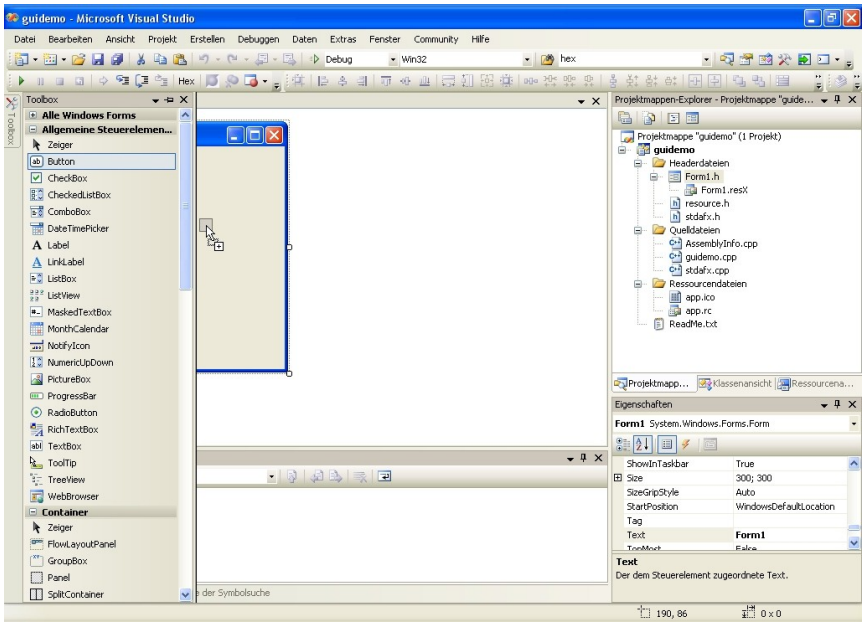
Hier beginnt die Klasse Form1. Sie endet direkt hinter der gleichfarbig unterlegten Zeile am Ende der Datei.

Hier beginnt der Konstruktor der Klasse Form1. Hier wird die Methode InitializeComponent() aufgerufen, durch die die Steuerelemente initialisiert werden. Hier können auch eigene Variablen initialisiert werden!

Das ist der Destruktor der Klasse Form1. Die zuvor dynamisch erzeugte Steuerelemente werden wieder zerstört. Hier kann auch selbst dynamisch reservierter Speicher frei gegeben werden!

In diesem Bereich werden die Steuerelemente mit den gewünschten Eigenschaften versehen.
Hier nichts ändern, wenn man nicht weiß, was man tut!

Stets zu Diensten



Ein leeres Fenster macht nun noch nicht viel Freude. Also müssen Steuerelemente, mit denen der Benutzer das Programm bedienen kann, auf der Form platziert werden. Das geht ganz einfach! In der Toolbox (der Werkzeugkasten) lassen sich die gewünschten Steuerelemente auswählen und mit gedrückter linker Maustaste auf der Form platzieren. Man kann die Größe und die Position und viele weitere Eigenschaften eines Steuerelementes verändern.

Die Toolbox ist aber nur bedienbar, wenn die Form angezeigt wird. Ist nur ein Codefenster sichtbar, bleibt der Werkzeugkasten zu!

Die drei wahrscheinlich am häufigsten benötigten Steuerelemente sind:

- die Befehlsschaltfläche (Button)
- die TextBox
- das Bezeichnungssteuerelement (Label)

Alle Steuerelemente reagieren auf so genannte Ereignisse. Ein Ereignis ist meist (aber nicht nur!) eine Benutzeraktion. Beispielsweise ist das Standard-Ereignis eines Buttons, dass man auf diesen klickt. In der Fensterprogrammierung werden nun diese Aktionen eines Benutzers an das betreffende Fenster weiter geleitet und das Fenster reagiert dann entsprechend -oder auch nicht!

Reaktionen erfolgen nur dann, wenn für ein Steuerelement für ein bestimmtes Ereignis wie Klick auf den Button eine Ereignisroutine vorliegt; man spricht auch von einem Eventhandler.

Wurden keine Eventhandler vorgesehen, passiert natürlich nichts! Bei der Benutzung eines Fensterprogramms werden permanent Ereignisse ausgelöst, ohne dass der Benutzer etwas davon merkt! Die meisten Ereignisse werden aber nicht „behandelt“, so dass für den Benutzer gar keine Reaktion sichtbar wird.

Hier noch ein paar Worte zu der CPP-Datei, die den Namen des Projektes trägt;
hier **`guidemo.cpp`**.

In dieser Datei befinden sich lediglich wenige Zeilen und hier muss auch nichts geändert werden.

Im Wesentlichen wird mit der vorletzten Zeile die Applikation gestartet und damit die Ereigniswarteschlange eingerichtet, wodurch Benutzeraktionen an das Fenster zur Behandlung weiter gereicht werden.

```
// guidemo.cpp: Hauptprojektdatei.
#include "stdafx.h"
#include "Form1.h"
using namespace guidemo;

[STAThreadAttribute]
int main(array<System::String ^> ^args)
{
    // Aktivieren visueller Effekte von Windows XP,
    // bevor Steuerelemente erstellt werden
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);

    // Hauptfenster erstellen und ausführen
    Application::Run(gcnew Form1());
    return 0;
}
```

Was wollt ihr denn?

Nein, kein Maoam! Aber weitere Steuerelemente natürlich! Ziehen Sie mit mir einen Button auf die noch leere Form. Sie können den Button nun nach eigenem Gusto positionieren und in der Größe verändern. Im Eigenschaftsfenster sind nun zuerst einmal zwei Eigenschaften von Bedeutung. Die **`Name`**-Eigenschaft und die **`Text`**-Eigenschaft.

Die **`Name`**-Eigenschaft ist besonders wichtig! Die **`Name`**-Eigenschaft ist die Bezeichnung des Steuerelementes innerhalb des Programms. Das ist gleichbedeutend mit der Benennung einer Variablen. Eigentlich ist es auch eine Variable, nämlich eine Variable vom Typ „Handle auf Button“.

Im Programmcode der Form1 (Datei **`Form1.h`**) hat sich nun folgendes geändert:

Das VS hat nun unterhalb des Destruktors eine solche ***Membervariable*** eingefügt. Der Zugriffbezeichner ist **`private`**. In der Funktion **`InitializeComponents()`** wird das Objekt erzeugt. Anschließend stattet VS das neu erzeugte Steuerelement mit den gewünschten Eigenschaften aus:

```
public ref class Form1 : public System::Windows::Forms::Form
{

//...anderer Code

private: System::Windows::Forms::Button^  btnKlickMich;

//...anderer Code

void InitializeComponent(void)
{
   this->button1 = (gcnew System::Windows::Forms::Button());
   this->SuspendLayout();
   //
   // button1
   //
   this->button1->Location = System::Drawing::Point(83, 155);
   this->button1->Name = L"btnKlickMich";
   this->button1->Size = System::Drawing::Size(75, 23);
   this->button1->TabIndex = 0;
   this->button1->Text = L"Klick Mich!";
   this->button1->UseVisualStyleBackColor = true;
```

Variable vom Typ Handle auf Button

dynamische Erzeugung des Objektes per Konstruktor

Festlegen der Objekteigenschaften

Jedes weitere Steuerelement erscheint als eine private Variable vom Typ „Handle auf Steuerelement“, wobei für Steuerelement der betreffende Typ einzusetzen ist. Der Typ „Handle auf“ wird durch das Caret **^**, das ist das kleine „Dach“, angegeben. Ein Handle kann man sich als eine eindeutige Bezeichnung für ein Datenobjekt vorstellen[1]. Das Handle wird vom Betriebssystem geliefert!

Vor dem Datentyp **`Button`** ist übrigens der Namespace angegeben, in dem der Datentyp zu suchen ist (hier **`System::Windows::Forms`**). Es wird ja mit .NET programmiert -und .NET ist nichts anderes, als eine riesige Klassenbibliothek, in der die einzelnen Klassen in verschiedenen Namespaces hierarchisch organisiert sind.

Schaut man sich den Programmcode zum Festlegen der Objekteigenschaften wie z.B. die **`Text`**-Eigenschaft an, stellt man fest, das der Zugriff über den Pfeil-Operator (**`->`**)erfolgt. Schließlich werden die Objekte ja dynamisch erzeugt! In diesem Zusammenhang bezeichnet **`this`** die Form selbst.

1 Insofern ist es kein Zeiger, hat aber in manchen Situation Ähnlichkeiten.

Reaktionen auf Aktionen

Jetzt befindet sich endlich mal ein Steuerelement auf der Form. Platzieren Sie gleich noch eines, nämlich eine **`TextBox`**. Benennen Sie die **`TextBox txtEingabe`**.

Sehen Sie sich nun auch den neu erzeugten Code an! Sie müssen wieder eine Variable vom Typ „Handle auf TextBox“ namens **`txtEingabe`** finden. Weiter unten dann die Erzeugung mit **`gcnew`** und zum Schluss die Initialisierung der Eigenschaften.

Anmerkung

In C++ werden dynamische Objekte mit **`new`** erzeugt. Bei .NET wurde ein sogenannter „***managed Code***“ eingeführt. Das bedeutet nichts anderes, als dass das System sich nun um die Verwaltung des reservierten Speichers kümmert. Objekte, die mit **`new`** erzeugt wurden, müssen vom Programmierer mit **`delete`** zerstört werden. Das entfällt bei **`gcnew`** für „managed Code“!

Werden jedoch eigene Objekte mit **`new`** erzeugt (es gibt Situationen, in denen das erforderlich sein kann!), dann müssen diese auch mit **`delete`** zerstört werden, damit der Speicher an das Betriebssystem zurück gegeben wird. Ansonsten entstehen sogenannte „***Memory leaks***“.

Zurück zu den Reaktionen!

Beim Klick auf den Button soll nun etwas passieren! Es soll der Text, der sich im Textfeld befindet, mit einer so genannten **`MessageBox`** angezeigt werden. Hierzu muss also ein Eventhandler erstellt werden.

Das geht ganz einfach: Doppelklick auf den Button, und schon landen wir im Codefenster. Dort wurde nun automatisch ein Eventhandler für das Ereignis „Klick auf Button **`btnKlickMich`**“ erstellt. In der Datei **`Form1.h`** taucht nun folgendes auf:

```
//Im Bereich InitializeComponent(void)

// btnKlickMich
...
this->btnKlickMich->Click += gcnew System::EventHandler(this, &Form1::btnKlickMich_Click);

...

//Am Ende der Datei, aber noch innerhalb der Klasse Form1
private: System::Void btnKlickMich_Click(System::Object^  sender, System::EventArgs^  e) {
         }
```

In der Klassenbeschreibung wird nun eine private Methode namens **`btnKlickMich_Click()`** mit zwei Parametern angegeben. Für diese Methode wird im Bereich der Initialisierung ein zugehöriger Eventhandler eingerichtet.

Wie muss man das verstehen? Das Button-Objekt hat eine ***Eigenschaft*** namens **`Click`**. Diese Eigenschaft nimmt die ***Adresse*** einer Methode/Funktion auf, die bei Eintreten des Ereignisses ausgeführt werden soll.

In der Methode **`btnKlickMich_Click()`** ist nun der Programmcode einzutragen, der bei Auslösen des Ereignisses auszuführen ist. Das ist nun gerade mal eine Anweisung, nämlich:

```
MessageBox::Show( txtEingabe->Text );
```

Erstellen und starten Sie Applikation. Tragen Sie einen Text in die TextBox ein und klicken Sie auf den Button.

Wie geht es weiter?

Dreh- und Angelpunkt jeder eigenen Windows Form Anwendung ist das Auseinandersetzen mit den Steuerelementen, die man für seine Applikation benötigt. Das Herausfinden der Eigenschaften und der „richtigen“ Ereignisse bei bestimmten Benutzeraktionen.

Wichtig ist noch, wie man eigene Klassen in ein Windows Form Projekt einbindet. Im Projekt Explorer fügt man wie auch bei Konsolenprogrammen die benötigten Header- und Codedateien hinzu.

Headerdateien von eigenen Klassen werden per **`include`** in der Datei **`stdafx.h`** eingebunden. Ein entsprechender Kommentar weist den Weg.

```
// stdafx.h : Includedatei für Standardsystem-Includedateien
// oder häufig verwendete projektspezifische Includedateien,
// die nur in unregelmäßigen Abständen geändert werden.
#pragma once

// TODO: Hier auf zusätzliche Header, die das Programm erfordert, verweisen.

//Für die Klasse Person
#include "Person.h"

//Um den Namespace std in Form1 einzubinden
using namespace std;
```

Um ein Objekt erzeugen zu können, muss, wie übrigens für jeden anderen grundlegenden Datentyp (z.B. **`int`**, **`double`** **`bool`** etc.) auch, eine Variable vom gewünschten Typ und Zugriffsbezeichner (**`private`**) deklariert werden.

```
...
//Meine Variablen
private:
   int iZahl;
   String ^sText;
   Person *myPerson;

private: System::Windows::Forms::Button^  btnKlickMich;
private: System::Windows::Forms::TextBox^  txtEingabe;

...
```

Die Deklaration erfolgt dort, wo auch das VS seine Objektvariablen für die Steuerelemente deklariert. Im Beispiel ist das direkt hinter dem Destruktor und vor den Steuerelementvariablen.

Die Initialisierung der Variablen erfolgt dann im Konstruktor.

In dem Beispielprogramm **`guidemo`** sind diese Vorgehensweisen noch kommentiert. Insbesondere die Umwandlung von Strings aus .NET in Strings der STL und umgekehrt ist beschrieben, da solche Strings noch oft in „alten Klassen“ vorkommen.

```
Form1(void)
{
   InitializeComponent();
   //
   //TODO: Konstruktorcode hier hinzufügen.
   //

   iZahl = 100;
   sText = "Hallo Welt";
   myPerson = new Person();
}
```