

MAXIMIZE THE
BUSINESS VALUE
OF SOFTWARE

.NET Komponententwicklung

Bernd Ua – <http://www.uaconsulting.de>

Borland®

Komponentenkonzepte

- ❖ Funktionalität
- ❖ Modularität
- ❖ Kapselung
- ❖ Wiederverwendbarkeit
- ❖ Austauschbarkeit

„Klassische“ Komponenten

- ❖ OCX / ActiveX Controls
- ❖ Java Beans
- ❖ VCL/CLX-Komponenten
- ❖ „OLE-Server“

.Net Komponenten

- ❖ Begriff der Komponente ist oft weiter gefasst
- ❖ Bezeichnet teilweise auch Anwendungen
- ❖ Im Gegensatz zu anderen Komponenten relativ „sprachunabhängig“
- ❖ Können über Sprachgrenzen hinweg vererbt werden

Kategorien

- ❖ Einfache Komponente
- ❖ Windows Steuerelement
- ❖ Web User Control
- ❖ Web Custom Control

Web User Control vs.

❖ Web User Control

- Wie eine Web Forms Seite, einfach zu erstellen, dynamisch übersetzt, Wiederverwendung durch Kopieren

❖ Web Custom Control

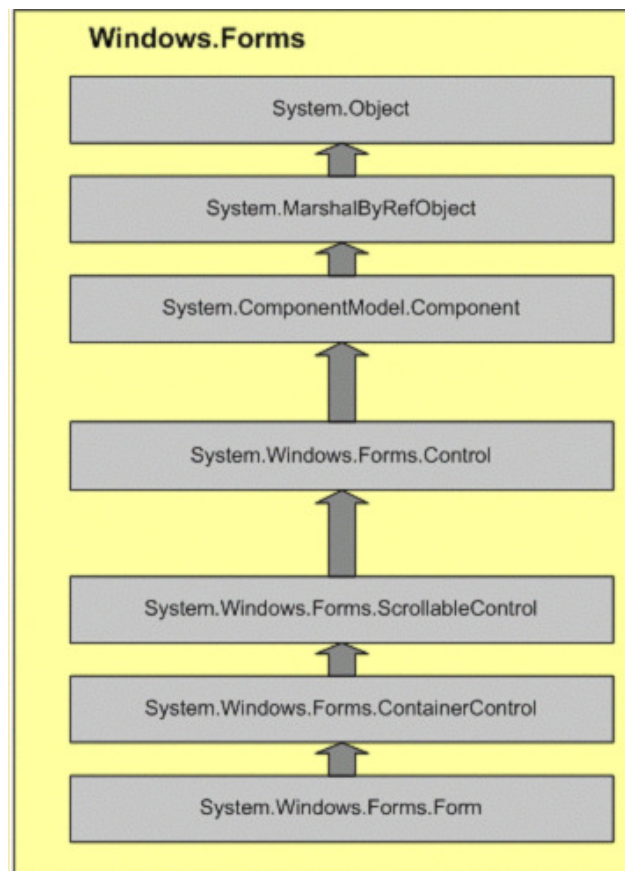
- Volle Design-time-Unterstützung (Toolbox, Eigenschafts-Fenster usw.), keine visuelle Oberfläche im Designer, GAC-fähig

Anforderungen

Komponenten müssen mehrere Schnittstellen bedienen:

- ❖ Die Schnittstelle zur IDE
- ❖ Die Schnittstelle zum Komponentenanwender
(public)
- ❖ Schnittstellen zum Komponententwickler
(protected)

Windows.Forms



Basisklassen für Komponenten

❖ Einfache Komponenten

- `System.ComponentModel.Component`

❖ Sichtbare Komponenten

- `System.Windows.Forms.Control`
- `System.Windows.Forms.ScrollableControl`
- `System.Windows.Forms.ContainerControl`

❖ Verbundkomponenten in .NET (Frames)

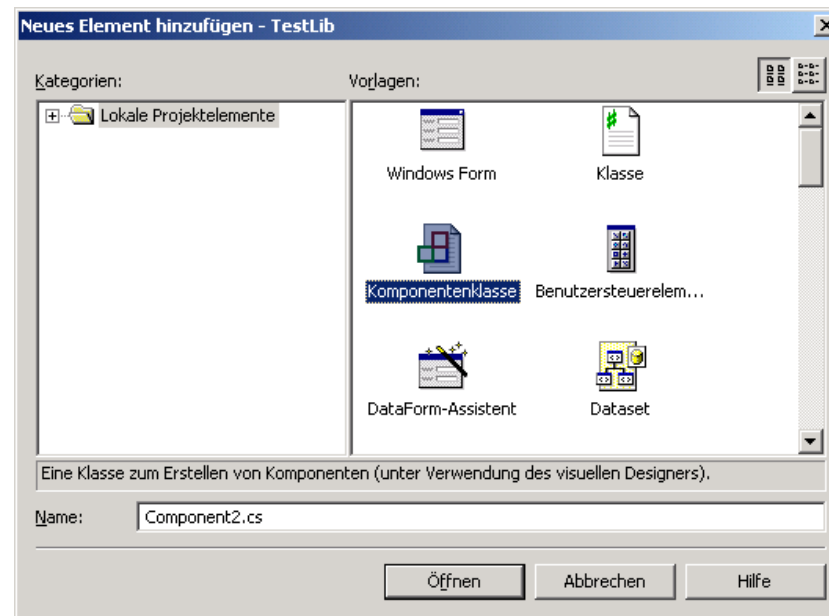
- `System.Windows.Forms.UserControl`

Komponenten erstellen

- ❖ Möglich direkt Quelltext des Formulars
- ❖ Sinnvoller in eigener Quelltextdatei und innerhalb einer Bibliothek

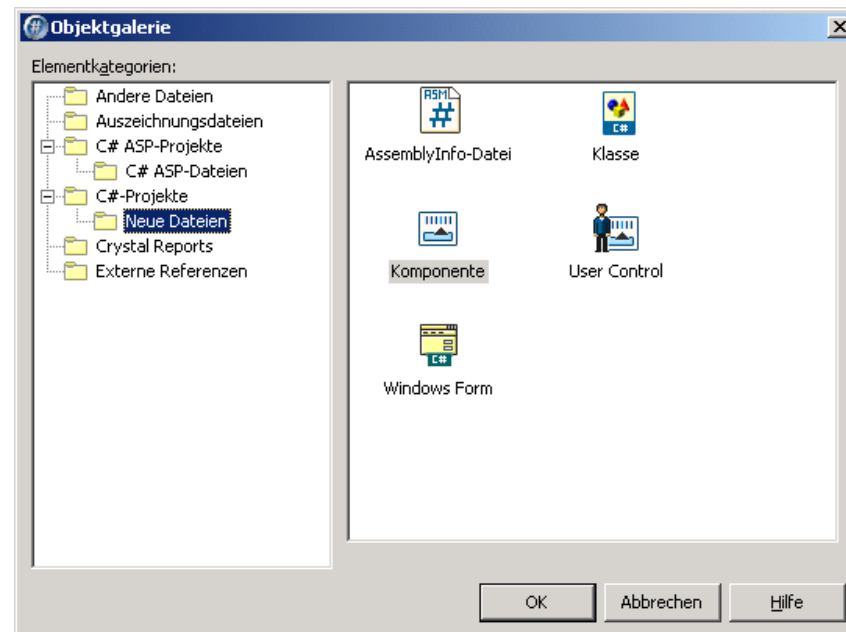
In Visual Studio

- ❖ Neues Projekt | Klassenbibliothek
- ❖ Klasse löschen
- ❖ Neues Element hinzufügen:
Komponenten-
klasse



In Delphi 2005

- ❖ Datei | Neu | C# Klassenbibliothek
- ❖ Klasse löschen
- ❖ Datei | Neu | Weitere | Komponente



Minimalkomponente

- ❖ Vorfahre von Component1 ist System.ComponentModel.Component

- ❖ Component1 weist zwei Konstruktoren auf

```
public Component1(System.ComponentModel.IContainer container)
    { ... }
public Component1()
    { ... }
```

- ❖ Der erste wird vom Designer verwendet und benötigt. Initialisierungscode muss ggfs. Zweimal vorliegen

System.ComponentModel. Component

- ❖ Basisklasse für Komponenten
- ❖ Wichtige Eigenschaften
- ❖ Container
 - Der Container in dem die Komponente ggfs. enthalten ist
- ❖ DesignMode (geschützt)
 - Gibt an ob die Komponente im Designer instanziiert ist
- ❖ Dispose/Disposed
 - Methode bzw. Ereignis zur Speicherfreigabe

Verwaltung

- ❖ Die Eigenschaften Container und Site geben bieten Zugriff auf die Verwaltungsschnittstellen der Komponente
- ❖ Container enthält die Schnittstelle des umgebenden Containers
- ❖ Site die Komponentenspezifische Verwaltungsschnittstelle ISite
- ❖ Der (eindeutige) Name einer Komponente wird durch die ISite-Schnittstelle verwaltet

Implementierte Schnittstellen

- ❖ Component implementiert :
- ❖ IDisposable
 - Dispose Methode für Freigabe
- ❖ IComponent
 - Ereignis Disposed
 - Eigenschaft Site

Component Dispose

- ❖ Dispose (`bool disposing`) von Component wird sowohl von Finalize (`disposing = false`) als auch von Dispose (`disposing true`) gerufen
- ❖ Zugriff auf andere Komponenten und Objekte darf nur bei `Disposing=true` erfolgen !

Freigabe von Containern/Formularen

- ❖ Formulare und Container rufen automatisch Dispose für enthaltene Komponenten
- ❖ Bei Formularen wird Dispose automatisch nach dem Schließen gerufen
- ❖ Aufräumarbeiten auf Komponentenebene durch Überschreiben bzw. erweitern von Dispose

System Windows Forms Control

- ❖ Implementiert Standardfunktionalität für Visualisierung
- ❖ Definiert die Begrenzungen(**Position** and **Size**)
- ❖ Liefert ein Fensterhandle (**Handle**)
- ❖ Liefert Zugriff auf
 - Tastaturereignisse
 - Zeichenereignisse
 - Mausereignisse

Vergleich von FCL und VCL

- ❖ Keine Entsprechung zu TGraphicControl
- ❖ Weniger Basisklassen (TCustomXXX) für Vererbung (XxxxBase)
- ❖ „Nur“ Control statt TCustomControl und TControl

Control Styles

❖ **TControl.ControlStyle** ersetzt durch **GetStyle/SetStyle**

```
SetStyle( ControlStyles.ResizeRedraw, True );  
SetStyle( ControlStyles.Opaque, True );  
// Double Buffering  
SetStyle( ControlStyles.UserPaint, True );  
SetStyle( ControlStyles.AllPaintingInWmPaint, True );  
SetStyle( ControlStyles.DoubleBuffer, True );
```

Benachrichtigungen

- ❖ Komponentenbenachrichtigungen erlauben Nachfahren auf Statusänderungen zu reagieren
- ❖ In der VCL erledigen das Nachrichtenhandler der Form für z.B. `CM_EnableChanged`
- ❖ Bei .NET reagiert die Komponente durch überschreiben von Methoden (z.B. `OnEnabledChanged`)
- ❖ .NET veröffentlicht in der Regel sogar öffentliche Ereignisse dafür (z.B. `EnabledChanged`)

Control vs TWinControl

- ❖ Zustand über boolesche Properties statt über Control/ComponentState, zB
 - Created
 - Focused
 - RecreatingHandle
 - IsDisposed

Eigenschaften

- ❖ In C# vereinbart durch sogenannte Accessoren
- ❖ Kapseln Get- und Set-Routinen ähnlich den Eigenschaften in Delphi
- ❖ Fehlende Get- oder Set-Methoden bewirken readonly oder writeonly Eigenschaften

Beispiel

```
private int m_MyProperty;
public int MyProperty
{
    get
    {
        return m_MyProperty;
    }
    set
    {
        if (value > 5)
        {
            MessageBox.Show("Ungültiger Wert");
        }
        else
        {
            m_MyProperty = value;
        }
    }
}
```

Die Delphi Version

```
private
    FMyProperty : Integer;
    ...
public
    function get_MyProperty : Integer;
    procedure set_MyProperty (Value : Integer)
    property MyProperty : Integer
        read get_MyProperty
        write set_MyProperty;
End;
```

Steuerung des Objektinspektors

- ❖ Kein Vergleichbare Direktive zu published in Delphi
- ❖ .NET verwendet Attribute zur Steuerung der Anzeige

```
[Browsable (true)]  
[Category( 'Darstellung' ) ]  
[Description( 'Firmenname für Aboutbox' ) ]  
[DefaultValue( typeof( string ), 'Your Company' ) ]  
property Company : String read get_Company  
write set_Company;
```

Attribute für Komponente

- ❖ **DefaultProperty**
 - Gibt die Standardeigenschaft an
- ❖ **DefaultEvent**
 - Gibt das Standardereignis an
- ❖ **ToolboxBitmap**
 - steuert das Designer-Bitmap für Komponente

Attribute für Eigenschaften

- ❖ **Browsable**
 - Attribut sichtbar im Objektinspektor
- ❖ **Description**
 - Beschreibung für Objektinspektor
- ❖ **Category**
 - Steuert die Kategorie im Objektinspektor
- ❖ **Defaultvalue**
 - Gibt den Standardwert für das Attribut an

Attribute für Eigenschaften

- ❖ **Localizable**
 - Gibt an ob die Eigenschaft lokalisiert werden soll
- ❖ **DesignOnlyAttribute**
 - Das Attribut steht nur zur Designzeit zur Verfügung
- ❖ **DesignerSerializationVisibility(DesignerSerializationVisibility.Content)**
 - Steuert die Serialisierung untergeordneter Komponenten

Besonderheiten von Delphi

- ❖ Für Eigenschaften im public-Bereich fügt Delphi automatisch [Browsable(false)] ein
- ❖ Für Eigenschaften im published-Bereich fügt Delphi automatisch [Browsable(true)] ein

Untergeordnete Komponenten

- ❖ Werden nicht Standardmässig serialisiert
- ❖ Serialisierung steuert das Attribut
DesignerSerializationVisibility der Eigenschaft

Untergeordnete Komponenten

- ❖ Eigene Klassen sind nicht standardmäßig im Objektinspektor aufklappbar
- ❖ Lösung:
- ❖ Attribut auf Klassenebene:
[TypeConverterAttribute(typeof(System.ComponentModel.ExpandableObjectConverter))]

Ereignisse

- ❖ Basieren in .NET auf Delegationen
- ❖ Werden als event deklariert
- ❖ Ermöglichen es bei bestimmten Aktionen Methoden aufzurufen, die erst zur Laufzeit bestimmt werden

Delegate

- ❖ Entsprechen in gewisser Weise Funktionspointern
- ❖ Mit Hilfe eines Delegates kann ein Verweis auf eine Methode in ein Delegatobjekt eingeschlossen werden
- ❖ das Delegatobjekt kann anschließend an Code übergeben werden, der wiederum die referenzierte Methode aufrufen kann
- ❖ Sind im Unterschied zu Funktionszeigern in C oder C++ typsicher

Delegaten verwenden

❖ Deklarieren eines Delegaten

```
public delegate void ProcessSomething (Object obj);
```

❖ Verwenden des Delegaten

```
public void DoProcessing (ProcessSomething processSomething )  
{  
    foreach (Object o in list)  
    {  
        processSomething(b);  
    }  
}
```

Delegaten II

- ❖ Ein Delegat wird instanziiert, indem er mit einer bestimmten Methode verknüpft wird
- ❖ Beim Konstruktoraufruf des Delegaten mit `new` wird eine Methode mit passender Signatur übergeben

```
static void PrintObj(Object obj)
    {
        Console.WriteLine(" {0}", obj.ToString());
    }
...
x.DoProcessing (new ProcessSomething(x.PrintObj));
```

Ereignisse

- ❖ Werden mit dem Schlüsselwort `event` definiert

```
public delegate void ChangedEventHandler  
    (object sender, EventArgs e);
```

```
...
```

```
public event ChangedEventHandler Changed;
```

- ❖ Werden mit dem Schlüsselwort `event` definiert

Ereignisse

- ❖ Delegat definiert Typ des Handlers:

```
public delegate int MyCallback(int a);
```

- ❖ Event schützt Zugriff auf Delegaten:

```
public event MyCallback TheEventHandler;
```

- ❖ Client fügt Delegat zu Event hinzu:

```
TheEventHandler += new MyCallback(...);
```

- ❖ Server ruft Handler auf:

```
if (TheEventHandler != null)  
    TheEventHandler(this, e);
```

Ereignisse in Delphi

- ❖ Analog anderen Eigenschaften
- ❖ Multicast verwenden !

```
// Multicast Event  
[Browsable (true)]  
property ValueChanged: EventHandler  
    add FValueChanged  
    remove FValueChanged;
```

Einschränkungen

- ❖ Ereignisse können ausschließlich in der Klasse aufgerufen werden, in der Sie deklariert wurden
- ❖ abgeleitete Klassen sind nicht in der Lage, in der Basisklasse deklarierte Ereignisse direkt aufzurufen
- ❖ Diese Möglichkeit muss durch eine separate (virtuelle) Methode in der abgeleiteten Klasse zur Verfügung gestellt werden

Grundlegende

- ❖ Toolbox/Objektinspektor
 - Design-Time-Attribute
- ❖ Eigenschafts-Fenster
 - Design-Time-Attribute
- ❖ Code-Serialisierung
 - Typ-Konverter

Typ-Konverter

- ❖ Konvertiert Werte in unterschiedliche Datentypen
- ❖ Hier: Eigenschaftswerte von und nach Zeichenketten für Eigenschaftsfenster
- ❖ Erspart den *UITypeEditor*
- ❖ Kann die Gültigkeit der Zeichenkette überprüfen.

TypeConverter definieren

- ❖ Eigene Klasse ableiten von `System.ComponentModel.TypeConverter`
- ❖ Klasse via `Attribute` für Eigenschaft angeben:
- ❖ `[TypeConverter(typeof(<Namespace>..<Klassenname >))]`

TypeConverter implementieren

❖ Überschreiben von

1. **CanConvertFrom**-Methode

- Gibt an, aus welchem Typ der Konverter konvertieren kann.

2. **ConvertFrom**-Methode

- Implementiert die Konvertierung

3. **CanConvertTo**-Methode

- Gibt an, in welchen Typ der Konverter konvertieren kann (nicht nötig für Zeichenfolgetypen)

4. **ConvertTo**-Methode

1. implementiert die Konvertierung

5. **IsValid**-Methode

1. führt die Überprüfung aus

UI-Typ-Editor

- ❖ Benutzerdefinierter Editor für Eigenschaftswert
- ❖ Entweder als eigenständiger Dialog oder Drop-down-Fenster
- ❖ Nutzt Designer Services
- ❖ Von *UITypeEditor* abgeleitet
- ❖ Wird über *Editor* Attribut verknüpft
- ❖ Kann vorhandene UI Editoren nutzen:
FileNameEditor, ColorEditor etc.

Ui-Typeditor implementieren

- ❖ Klasse ableiten von `System.Drawing.Design.UITypeEditor`
- ❖ Methode `GetEditStyle`-Methode überschreiben um den Stil anzugeben (modal,dropdown,none)
- ❖ Methode `EditValue` überschreiben

Designer

- ❖ Modifiziert das Design-Time-Verhalten der Komponente
- ❖ IDesigner-Interface, ComponentDesigner, ControlDesigner
- ❖ Filtert die Eigenschaften für das Eigenschaftsfenster.
- ❖ Kann neue Eigenschaften hinzufügen.
- ❖ Fügt Menübefehle für die Komponente hinzu.

Designer implementieren

- ❖ Eigen Klasse ableiten von
- ❖ Für Component-Nachfahren :
 - `System.ComponentModel.Design.ComponentDesigner`
- ❖ Für Control-Nachfahren :
 - `System.Windows.Forms.Design.ControlDesigner`
- ❖ Referenz auf `System.Design` zufügen !

Designer implementieren

- ❖ Methode Verbs überschreiben und neue Kontextmenüeinträge zurückliefern
- ❖ Ereignishandler für die jeweiligen Verben deklarieren

Weiterführende Themen

- ❖ Extender Provider
- ❖ Verteilen & Lizenzieren

Fragen

❖ Bernd.Ua@uaconsulting.de