

MAXIMIZE THE
BUSINESS VALUE
OF SOFTWARE

.NET Component Development

Bernd Ua

Borland®

What different ?

- ❖ .Net Components
- ❖ can be used cross language
- ❖ can be inherited cross language
- ❖ You should stay to CLS Compliancy

Categories of Components

- ❖ Simple Components
- ❖ Windows Controls
- ❖ Web User Control
- ❖ Web Custom Control

Web User Control vs.

❖ Web User Control

- Can be designed like a website
- Simple to create
- Compiled dynamically
- Bound to language of control
- Poor Design-Time Support

❖ Web Custom Control

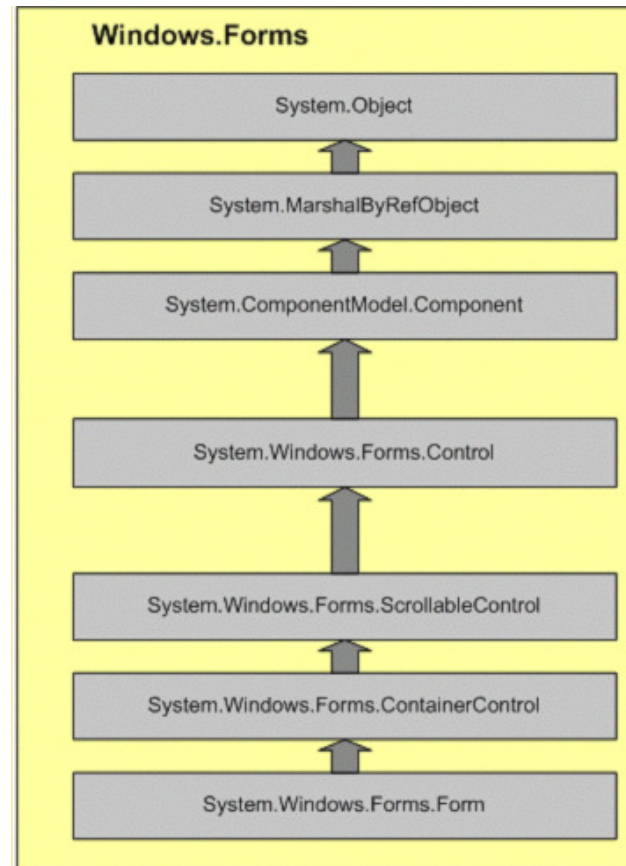
- Full Design-Time Support (Toolbox, Properties)
- No visual Design
- Can be used from GAC

Requirements for Components

Components have to serve several Interfaces:

- ❖ An Interface to the Development Environment
- ❖ An Interface for the user of this Component
- ❖ An Interface for other Component Developers

Windows.Forms Hierarchy



Base Classes

❖ Simple Components

- `System.ComponentModel.Component`

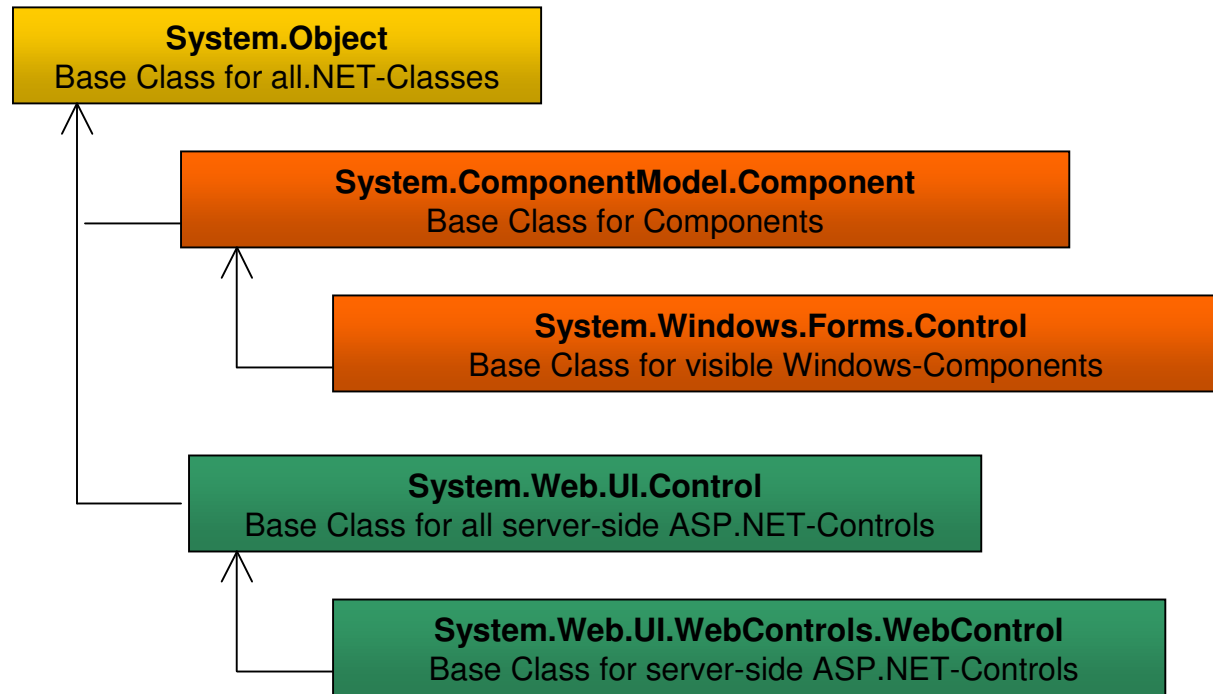
❖ Visible Components

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

❖ Compound Components in .NET (Frames)

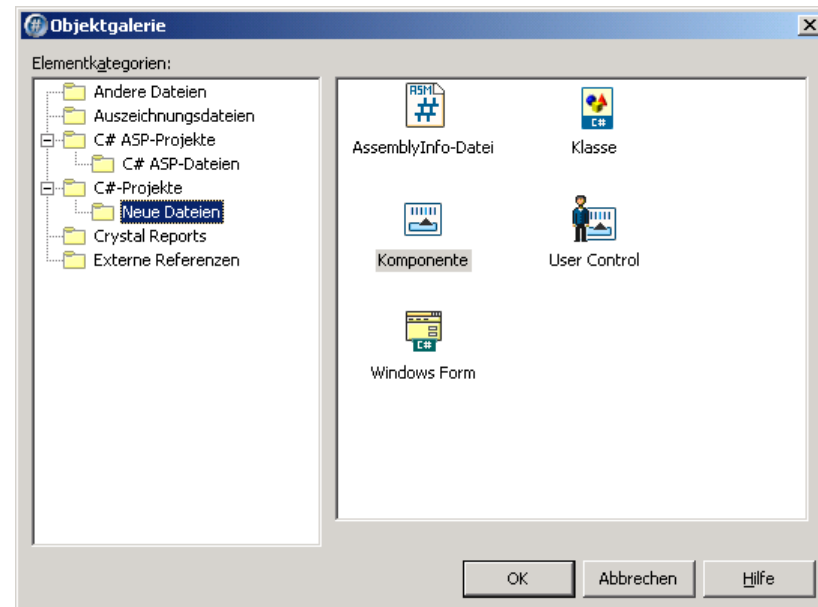
- `System.Windows.Forms.UserControl`

Base Classes Hierarchy



Creating Components in Delphi 2005

- ❖ C#
- ❖ File | New | Other | C# Projects | Class Library
- ❖ (Delete empty Class)
- ❖ File | New | Other | C# Files | Component
- ❖ Delphi
- ❖ File | New | Other | Delphi Projects | Package
- ❖ File | New | Other | Delphi Files | Component



A minimal Component

- ❖ Ancestor von Component1 is
System.ComponentModel.Component

- ❖ usually in C# there are two constructors

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

- ❖ First one is used by the Designer. Code for initializing your component has to be there two times

System.ComponentModel. Component

- ❖ Base class for Components
- ❖ Important properties
- ❖ Container
 - A reference to the Container of the Component
- ❖ DesignMode (protected)
 - DesignMode is true if the component has been instantiated within the designer
- ❖ Dispose/Disposed
 - Method and Event when Component Resources are freed

Managing the Component

- ❖ Properties Container und Site give access to manage the component
- ❖ Property Container provides a reference to the Container of the Component
- ❖ Property Site is Component-Specific Interface managing for example the name of the component

Implemented Interfaces

- ❖ Component implements at least two interfaces :
- ❖ **IDisposable**
 - Contains Method Dispose for resource disposal
- ❖ **IComponent**
 - event Disposed raised when Dispose is called
 - property Site for the management interface

Component Dispose

- ❖ The method Dispose (`bool disposing`) is called within the Context of a finalize call (`disposing = false`) or within the context of a call to dispose (`disposing = true`)
- ❖ Remember to not use references to other objects and components within the context of finalize

Disposal of Containers and Forms

- ❖ Forms and Containers automatically call Dispose for any contained component
- ❖ Forms will call Dispose when they are closed if they are displayed in modal way
- ❖ Cleanup Code can be done overriding Dispose

System Windows Forms Control

- ❖ Implements functionality for visualization and window management
- ❖ Defines **Position** and **Size**
- ❖ Provides a windows handle
- ❖ Provides access to
 - Key events
 - Paint Events
 - Mouse Events

Comparison of FCL und VCL

- ❖ There is no TGraphicControl in FCL
- ❖ Less base classes of form TCustomXXX for inheritance
- ❖ „Only“ Control as a base class instead of TCustomControl and TControl

Control Styles

- ❖ **TControl.ControlStyle** is replaced by **GetStyle/SetStyle**

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

- ❖ Some but not all of this Styles have corresponding public properties, i.e. ResizeRedraw

Notifications

- ❖ Component Notification allow descendants to act on state changes
- ❖ Within the VCL this is done with message handlers like `CM_EnabledChanged`
- ❖ Within the FCL you handle this State changes, overriding the corresponding methods like `OnEnabledChanged`
- ❖ Usually .NET Components publish Events for this state changes like `EnabledChanged`.
- ❖ As a component developer you should prefer to override methods instead of providing event handlers, since you have more control concerning the order of calls then

Control versus TWinControl

- ❖ Current State can be accessed via boolean properties rather than Control or ComponentState
 - Created
 - Focused
 - RecreatingHandle
 - IsDisposed

Properties

- ❖ In C# these are implemented with accessor methods
- ❖ The methods encapsulate the getters and setters similar to a Delphi Property
- ❖ missing Getters- or Setters-Methoden result in readonly or writeonly properties like in Delphi

Sample

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

The 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;
```

Controlling the inspector

- ❖ There is no published directive in .NET
- ❖ .NET uses Attributes controlling the display in object inspector

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

Attributes for Components

- ❖ **DefaultProperty**
 - The default property of this component
- ❖ **DefaultEvent**
 - The default event of this component
- ❖ **ToolboxBitmap**
 - Describes the Designer-Bitmap used for this component

Attributes for Properties

- ❖ **Browsable**
 - Attribute is visible in Object Inspector
- ❖ **Description**
 - A description shown in the Object Inspector
- ❖ **Category**
 - Category in the Object Inspector
- ❖ **Defaultvalue**
 - Provides a default value for a property

Attributes for Properties (cont.)

- ❖ **Localizable**
 - Shows whether this property can be localized
- ❖ **DesignOnlyAttribute**
 - The Property is only accessible at design time
- ❖ **DesignerSerializationVisibility(DesignerSerializationVisibility.Content)**
 - Controls the Serialization of contained Components

Special Issues with Delphi Language

- ❖ The Delphi Compiler sets the `Browsable` Attribute to false (`[Browsable(false)]`) for any property with a visibility of `public`
- ❖ The Delphi Compiler sets the `Browsable` Attribute to true (`[Browsable(true)]`) for any property with a visibility of `published`

Contained Components

- ❖ They won't be serialized out of the box
- ❖ Properties have to be marked with `Attribute DesignerSerializationVisibility` set to `true`

Contained Components (cont.)

- ❖ Contained components can not be opened in object inspector
- ❖ Solution:
- ❖ You have to set an `Attribute` on property-level:
[`TypeConverterAttribute(typeof(System.ComponentModel.ExpandableObjectConverter))`]

Events in .NET

- ❖ Are based on Delegates
- ❖ Are declared with event
- ❖ A delegate can be seen as a type safe function pointer
- ❖ A delegate encapsulates a reference to a method and can be used in code to call the methods

Using Delegates

❖ Declaring a Delegate

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

❖ Using a Delegate

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

Delegates (cont.)

- ❖ A delegate is instantiated with a reference to a method

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

Events

❖ Declared with event

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

Calling Events

- ❖ The Delegate defines the Type of the Handler:

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

- ❖ Event protects the Delegate from external access:

```
public event MyCallback TheEventHandler;
```

- ❖ A Client adds a Delegate on an Event:

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

- ❖ Server calls the Handler:

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

Events in Delphi

- ❖ Are declared like other properties
- ❖ With FCL Components Multicast Syntax should be used
!

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

Constraints

- ❖ Events can only be called in the class they have been declared
- ❖ No descendant can call this event directly
- ❖ The developer has to provide a separate virtual method to call the event

Basic IDE-Integration

- ❖ **Toolbox/Object Inspector**
 - Design-Time-Attribute
- ❖ **Property-Window**
 - Design-Time-Attribute
- ❖ **Code-Serialization**
 - Type-Converter

Type-Converter

- ❖ Converts Values to different Data Types
- ❖ In this case : Property Values to and from String to display in Object Inspector
- ❖ Can validate String entered for a property

Declaring TypeConverters

- ❖ Inherits from `System.ComponentModel.TypeConverter`
- ❖ The `TypeConverter` is supplied via an `Attribute` for the Target Property
- ❖ `[TypeConverter(typeof(<Namespace>..<Klassenname >))]`

Implementing TypeConverters

❖ Override one or more of the following

1. **CanConvertFrom**-Method

- Provides the Types, the Converter can convert from

2. **ConvertFrom**-Method

- Implements the conversion

3. **CanConvertTo**-Method

- Provides the Type the Converter can convert to (not necessary for String-Type)

4. **ConvertTo**-Method

- Implements the conversion

5. **IsValid**-Method

- Validates the contents

UI-Typ-Editor

- ❖ Is a user-Defined-Editor for Property Values
- ❖ Can be a dialog or a drop-down-Window in object inspector
- ❖ uses Designer Services
- ❖ Inherits from *UITypeEditor*
- ❖ Is tied to a Property via *Editor* Attribute
- ❖ Can use existing UI Editors like *FileNameEditor*, *ColorEditor*

Implementing UI-Editors

- ❖ Inherit from `System.Drawing.Design.UITypeEditor`
- ❖ Override method `GetEditStyle` to provide the style of the Editor (modal,dropdown,none)
- ❖ Override Method `EditValue`

Designer

- ❖ Modifies the components behavior at Design Time
- ❖ IDesigner-Interface, implemented by ComponentDesigner, ControlDesigner
- ❖ Filters properties for the designer window
- ❖ Can provide new properties
- ❖ Can add new menu commands for a component

Implementing Designers

- ❖ Inherit a class from
- ❖ for Component-Descendants :
 - `System.ComponentModel.Design.ComponentDesigner`
- ❖ for Control-Descendants :
 - `System.Windows.Forms.Design.ControlDesigner`
- ❖ You'll need a reference to namespace `System.Design` !

Implementing Designers (cont.)

- ❖ Override Method Verbs and add new context menu commands for the result
- ❖ Implement Eventhandlers for the new verbs you added