

# The .NET Property Grid

Tutorial

Thomas Becker

Michael Voigt

# Topics

- Usage of PropertyGrid
- Visual Aspects of Properties
- Complex Properties
- Custom User Interface
- Customizing Properties
- Localizing Properties

# Introduction

- `System.Windows.Forms.PropertyGrid`
- wellknown from the VS Designer Environment
- Use `Add/Remove Items....` to add `PropertyGrid` control to your VS toolbar

# Using out of the box

- easy use
- operates via **Reflection**
- support for most of .NET classes
- use **SelectedObject** to display properties of passed object
- use **SelectedObjects** to display properties that are common to all the objects that are in the array of passed objects

# Sample 1 – Usage

# Why Customizing ?

- not very flexible out of the box
- problem is "beautifying" the contents of the grid
- standard behavior: display actual name of the property from the code
- if used as part of an interface for end-users, needs customizing (f.e. displaying „Employee Identification“ instead of „UserID“)

# Bend PropertyGrid to your Will

- **Attributes** : control Visual Aspects of Properties
- **TypeConverter** : converting types of values to other types, accessing standard values and subproperties
- **UITypeEditor** : base class to design value editors that can provide a user interface for representing and editing of Property values
- **ICustomTypeDescriptor** : provides an interface that supplies custom type information for an object

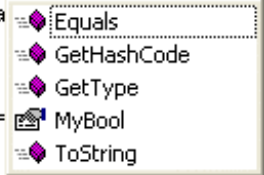
# Changing Visual Aspects

- **DefaultPropertyAttribute** : default property for the class, gets the focus first when the class is selected in the grid
- **CategoryAttribute** : category that the property is grouped by in the grid. default category is *Misc*
- **DescriptionAttribute** : displayed text in the description help pane below the properties, useful way to provide help text for the active property
- **ReadOnlyAttribute** : useful to keep a property from being editable in the grid
- **DefaultValueAttribute** : property's default value, values different from the default are displayed bold in the grid
- **BrowsableAttribute\*** : useful to hide a property from the grid. By default, a public property is always shown in the grid
- **RefreshPropertiesAttribute** : Indicates how to refresh when the associated property value changes

# Changing Visual Aspects II

- **BrowsableAttribute\*** : useful to hide a property from the grid. By default, a public property is always shown in the grid
- **EditorBrowsable** : Indicates how Editors like Intellisense see properties. **Does not effect the Property-Grid!** (see Sample 7)

```
// EditorBrowsable Attribute specifies how a property or method is viewable in an editor.  
// Nevertheless you can access the property, even it is hidid by EditorBrowsableState.Never  
// EditorBrowsable Attribute works only for classes, which are situated in another assembly.  
// If the class belongs to the same assembly, the property is nevertheless visible.  
// try to use intellisense to expand properties of myClass !  
MyClass myClass = new MyClass();  
bool myBool = myClass.  
bool myInvertedBool = myClass.  
  
// EditorBrowsable Attribute  
propertyGrid.SelectedObject =  
}
```



The image shows a screenshot of a code editor with a dropdown menu open over the line `bool myInvertedBool = myClass.`. The dropdown menu lists several methods: `Equals`, `GetHashCode`, `GetType`, `MyBool`, and `ToString`. The `MyBool` option is highlighted with a mouse cursor. The code is in C# and includes comments about the `EditorBrowsable` attribute and its effect on the property grid.

# Sample 2 – Visual Aspects

# Changing Visual Aspects

- Read only in runtime
- Only const expression, no code
- Not the mechanism for localisation
- Localizable-Attribute??

# Displaying Complex Properties

- **Default Behaviour** : the subproperties of your complex properties cannot be expanded
- **TypeConverter** : **unified** way of converting types of values to other types, as well as for accessing standard values and subproperties
- **ExpandableObjectConverter** : **easier** way of providing a type converter to convert expandable objects to and from various other representations
- **ConvertTo(...)** : implement the convert methods from your property values to string and back
- **RefreshProperties(RefreshProperties.Repaint)**: use refresh property attribute for subproperties to force repaint of string display

# Sample 3 – Complex Properties

# Custom Dropdown Pick Lists

- provide a `TypeConverter` for the Property, use existing derived converter like `StringConverter` for string properties
- override **`GetStandardValuesSupported()`**
- override **`GetStandardValues()`**
- override **`GetStandardValuesExclusive()`**, return **`false`**, if user should be able to type in additional values, otherwise **`true`**

# Sample 4 – Custom Picklist

# Custom User Interface

- Derive from `UITypeEditor`
- `GetEditStyle`
  - `DropDown` - `IWindowsFormsEditorService`
  - `Modal`
- Implement `UserControl`

# Sample 5 – Custom UI

# Localization & Customizing

- No OOTB feature
- Custom Reflection - `ICustomTypeDescriptor`
- Generate Custom `PropertyDescriptorCollection`
- Load Localization from Resource
- Enables conditional „Layout“

# Sample 6 – Dynamic Properties

# Conclusion

- I like PropertyGrid
- RAD OOTB
- Real World UI needs Customizing
- Why is Localization missing OOTB in .NET 1.1? What will come with 2.0?

# Links

- <http://www.c-sharpcorner.com/Code/2002/April/GlobalizedPropGrid.asp>
- <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/usingpropgrid.asp>
- <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconimplementingtypeconverter.asp>
- [http://www.codeproject.com/cs/miscctrl/bending\\_property.asp](http://www.codeproject.com/cs/miscctrl/bending_property.asp)
- <http://www.codeproject.com/cs/miscctrl/customizingcollectiondata.asp?target=propertygrid>
- <http://www.codeproject.com/cs/miscctrl/globalizedpropertygrid.asp?target=propertygrid>

**Still any Questions!?**

