Creating Custom User Controls : Basic – II

Source	http://www.dotnetspider.com/kb/Article2886.aspx

Categories of User Controls

Depending on the way the control draws itself, there are three kinds of custom controls:

- Non custom drawn: These are controls that use other controls' user interfaces to cover their own interface. An example is a Toolbar control that uses toolbar buttons to cover its interface.
- Custom drawn: These controls paint their user interface themselves according to input data, mouse and keyboard events, focus, and other variables. As an example, a PieChart control is custom drawn.
- Mixed: The mixed controls use both of the above methods to cover their user interface. For example, a Chart control with scrollbars is in this category.

What Are Custom Controls Made Of?

To implement custom controls we need to understand how they and their component parts work.
We have to learn about the visible and invisible parts of controls. Custom controls are made of
two main parts. The first part is the "black box". This part is private to the control and holds the
private data members and methods that build up the control's internal functionality. The second
part is the control's public interface. This interface is made up of public properties, events, and
methods. They expose the control's functionality allowing the code that uses the control to manipulate the control programmatically.

Technically, a control is a class derived from the base System.Windows.Forms.Control class. It
contains the basic functionality of any control, such as functionality for handling mouse events,
keyboard events, focus and paint events, preset styles, and properties. The most basic definition of
a custom control is as shown below:

```
public class MyControl:Control
{
}
```

We have to first learn the basic components of a Control class. It is important to

know and understand what these components are, and how to use them to implement control
functionality as they will be present in any control we create. These components make up the body
of the control's class, and represent the changes you implement into your custom control, on top of
the base functionality you inherit from the base Control class. In other words, we inherit from the
Control class some basic features, common for all controls, and we build custom functionality for
our control by adding these components. We could also modify an existing control, to add an
extra feature.

Private Fields

A private field is, as its name suggests, a field that cannot be accessed from the outside. When
building a custom control, the "outside" is the application that uses this control (it can also be
another custom control that uses your control). Usually, for every public property of the control,
there is at least one private field that stores the data exposed by it.

A good programming practice is to declare private class fields, and then expose them through
public properties (explained next).

Here's a code snippet that shows the definition of a control named MyControl, having four private fields:

```
public class MyControl : Control
{
private Color backgroundColor;
private Color foregroundColor;
private int intemCount;
private Brush backBrush;
}
```

Properties

When you select a control in the Form designer of Visual C# Express or Visual Studio, you can
see the control's properties in the Properties window. A property is an attribute associated with a
class or an object. For example, a common button has lots of properties: name, text, font, size, and
many others. All these properties exposed by a common button are shown in the Properties window

Properties are the key features of any control as they expose the control's settings and

data. The
public properties represent the way the user interacts with the settings of a control, by controlling
the way the user gets or sets the private fields that hold the settings and data.

Properties contain code that filters the data that is read or set, in their get and set accessors.
These accessors usually read or set the values of private members, which contain the actual data,
on behalf of the property. By defining only the get accessor of a property you make it read-only,
and by defining only the set accessor you make it write-only.

```
A property's default structure is:
public "type" "PropertyName"
{
 get
 {
 return "fieldName";
 }
 set
 {
 "fieldName" = value;
 }
}
```

Here, "type" represents the data type of the property (such as string), "PropertyName" is the
name of the property (such as BackgroundColor), and "fieldName" is the private field that stores
the property data. Note that the property itself doesn't contain any data, and it's free to set or return
any values in its get and set accessors.

Summary

Categories of User Controls
What Are Custom Controls Made Of?
Private Fields
Properties

***~~~ End of Article ~~~***