How to: Build Components

To build components for the 'ElectronicParts' program, your component needs a reference to the interfaces located in Shared.dll.

Important: Add [Serializable] attribute above every class you create (to enable to save the component) and provide a parameterless constructor as well.

1. Create a new class and implement '**IDisplayableNode'** interface of 'Shared'.

```
/// <summary>
/// Represents an <see cref="AndGate"/> with one output pin as boolean.
/// </summary>
[Serializable]
5 references | PeterHelf, 4 hours ago | 3 authors, 8 changes
public class AndGate : IDisplayableNode
{
```

2. In the constructor of your component initialize two ICollection<IPin> (e.g. List<IPin>) and add as many IPins (see next) as desired for input and output pins.

```
/// <summary>
/// Initializes a new instance of the <see cref="AndGate"/> class with two input pins and one output pin.
/// </summary>
1 reference | roman, 6 days ago | 1 author, 3 changes
public AndGate()
{
    this.Inputs = new List<IPin>() { new Pin<bool>(), new Pin<bool>() };

    this.Outputs = new List<IPin>() { new Pin<bool>() };
}
```

3. To create a new IPin, create a new class (e.g. Pin<T>) and implement the generic IPinGeneric<T> interface, where 'T' stands for the type of value the pin expects (input pins) or creates (output pins). For example, a simple 'and gate' will only expect and create bools, so create Pin<bool> as input and output pins.
   See and/or use 'ExamplePin' in the Shared.dll.

```
/// <summary>
/// An example implementation of the <see cref="IPinGeneric{T}"/> interface.
/// </summary>
/// <typeparam name="T">The type of the value.</typeparam>
[Serializable]
2 references | PeterHelf, 3 hours ago | 1 author, 1 change
public class ExamplePin<T> : IPinGeneric<T>
{
    /// <summary>
    /// Initializes a new instance of the <see cref="ExamplePin{T}"/> class.
    /// </summary>
    0 references | PeterHelf, 3 hours ago | 1 author, 1 change
    public ExamplePin()
    {
        this.Value = new ExampleValue<T>();
    }
```

4. Further there are 2 ‘Value’ properties. One is from the IPinGeneric<T> interface and one from its parent interface IPin. The first one you have to initialize in the constructor (e.g. this.Value = new Value<T>(); see next). The second one just has to return (get) and set the first value. You will need some casting in the set method of this property.

```
/// <summary>
/// Gets or sets the value of the pin.
/// </summary>
/// <value>The value of the pin.</value>
10 references | PeterHelf, 3 hours ago | 1 author, 1 change
public IValueGeneric<T> Value { get; set; }

/// <summary>
/// Gets or sets the value of the pin.
/// </summary>
/// <value>The value of the pin.</value>
75 references | PeterHelf, 3 hours ago | 1 author, 1 change
IValue IPin.Value
{
    get
    {
        return this.Value;
    }

    set
    {
        try
        {
            this.Value = (IValueGeneric<T>)value;
        }
        catch (InvalidCastException e)
        {
            this.Value = new ExampleValue<T>();
            Debug.WriteLine(e.Message);
        }
    }
}
```

5. Create a Value<T> class implementing the generic IValueGeneric<T> where T is the type of the value (e.g. Value<bool>). Implementing this interface will again provide two similar properties named 'Current'. One is from IValueGeneric<T> and one is inherited from its parent interface IValue. The IValue.Current (which is of type object) should only return (get) and set the IValueGeneric<T>.Current (which is of type T, e.g. bool).
   See and/or use 'ExampleValue' in the Shared.dll.

```
/// <summary>
/// An example implementation of the <see cref="IValueGeneric{T}"/> interface.
/// </summary>
/// <typeparam name="T">The type of the value.</typeparam>
[Serializable]
2 references | PeterHelf, 3 hours ago | 1 author, 1 change
public class ExampleValue<T> : IValueGeneric<T>
{
    /// <summary>
    /// Gets or sets the current value.
    /// </summary>
    /// <value>The current value.</value>
    9 references | PeterHelf, 3 hours ago | 1 author, 1 change
    public T Current { get; set; }

    /// <summary>
    /// Gets or sets the current value.
    /// </summary>
    /// <value>The current value.</value>
    68 references | PeterHelf, 3 hours ago | 1 author, 1 change
    object IValue.Current
    {
        get
        {
            return this.Current;
        }

        set
        {
            this.Current = (T)value;
        }
    }
}
```

6. **Execute-Method**: The execute method gets invoked in every frame. We recommend in this method to just check the values of the input pin(s) and update the value of the output pin(s). **Important**: Take care of the type of inputs because the user could add pins with other types. To ensure correct behavior of your component we recommend to just ignore unexpected input types and just use the pins with the expected input type.

7. **Activate-Method**: The Activate method gets invoked when the user double-clicks your component. You can use this method for example to update the picture of your component (e.g. power-on and power-off)

8. You can also provide a (or more) picture(s) for your component. Just use the 'Picture' property of the IDisplayableNode interface. If you change the picture in Activate- or Execute-method invoke the 'PictureChanged' event to notify the view that the current picture has changed.