Creating Custom Form Functions in the CMS Console UI

2017-02-14

Orckestra, Europe
Nygårdsvej 16
DK-2100 Copenhagen
Phone +45 3915 7600

www.orckestra.com

Contents

1 What are Form Functions?

The C1 CMS Forms system – the system building up administrative UI forms – is based on an open plug-in architecture allowing developers to add new Form UI Controls and Form Functions.

Form UI Control plug-ins deliver concrete UI elements, while Form Functions deliver objects typically used to feed a Form UI Control, like this:

```
<TextBox>
  <TextBox.Label>
    <DemoFunction FirstName="John" LastName="Doe"
xmlns="http://www.mycompany.net/ns/c1/formfunctions" />
  </TextBox.Label>
</TextBox>
```

In situations you would like to “inject" data – typically to populate a select box with specific options – you can use Form Functions to achieve this result.

1.1 Creating your Form Functions

To create your own functions you should create a std. C# Class Library project containing one or more classes (one per new function you require) that implement the interface Composite.Forms.Plugins.FunctionFactory.IFormFunctionFactory.

Here is a sample implementation:

```
[ConfigurationElementType(typeof(NonConfigurableFunctionFactory))]
public sealed class DemoFunctionFactory : IFormFunctionFactory
{
    public IFormFunction CreateFunction()
    {
        return new DemoFunction();
    }
}
```

This is a factory class, which build up your actual Form Function class. The Form Function classes returned from your factory class must implement Composite.Forms.Plugins.FunctionFactory.IFormFunction like this:

```
public sealed class DemoFunction : IFormFunction
{
    [FormsProperty()]
    [RequiredValue()]
    public string LastName { get; set; }

    [FormsProperty()]
    public string FirstName { get; set; }

    public object Execute()
    {
        return string.Format("Hello there {0} {1}", this.FirstName,
this.LastName);
    }
}
```

The Form Function class contains an Execute() method which is invoked when the Form Function should deliver a result. Before Execute() is invoked C1 CMS has moved all values from the Form markup to the respective properties (in this example ‘LastName’ and ‘FirstName’) and C1 CMS has also ensured that required values have been set.

The attribute “FormsProperty" defines that the property can be written to via Forms markup. The “RequiredValue" property defines that the property must be set before Execute() can be invoked.

The type of your properties can be any type. In this sample only string is used, but you can specify types like Guid, int etc.

The Form markup below will set the “FirstName" property on the Form Function object to “John" while the “LastName" property will be set to a value read from the Forms binding values.

```
<DemoFunction FirstName="John"
xmlns="http://www.mycompany.net/ns/c1/formfunctions">
  <DemoFunction.LastName>
    <read source="SomeBindingName"
xmlns="http://www.composite.net/ns/management/bindingforms/1.0" />
  </DemoFunction.LastName>
</DemoFunction>
```

1.2 Registering your Form Function

In order to use your new Form Function in C1 CMS’s Form markup it must be registered and assigned an XML namespace. To do so, do the following (substitute namespaces and type names with your own):

1. Copy the DLL containing your Form Function classes to the websites /bin/ folder

2. Edit the file /App_Data/Composite/Composite.config

3. Locate the element /configuration/Composite.Forms.Plugins.ProducerMediatorConfiguration/Mediators

4. Add a new child element like this (change the namespace to your own):

```
<add
type="Composite.Forms.StandardProducerMediators.FunctionProducerMediator,
Composite" name="http://www.mycompany.net/ns/c1/formfunctions" />
```

5. Locate the element /configuration/Composite.Forms.Plugins.FunctionFactoryConfiguration/Namespaces

6. Add a new child element like this (change the namespace, type and tag name to your own):

```
<Namespace name="http://www.mycompany.net/ns/c1/formfunctions">
  <Factories>
    <add type="FormFunctionDemo.DemoFunctionFactory, FormFunctionDemo"
name="DemoFunction" />
  </Factories>
</Namespace>
```

7. Restart the ASP.NET application by recycling the sites application pool (AppPool) or editing web.config.

The steps shown above will register the Function Factory class named **FormFunctionDemo.DemoFunctionFactory** as a new Form markup tag named "DemoFunction" belonging to the namespace http://www.mycompany.net/ns/c1/formfunctions

At this point you can execute your function using Form markup like this:

```
<DemoFunction FirstName="John"
xmlns="http://www.mycompany.net/ns/c1/formfunctions">
  <DemoFunction.LastName>
    <read source="SomeBindingName"
xmlns="http://www.composite.net/ns/management/bindingforms/1.0" />
  </DemoFunction.LastName>
</DemoFunction>
```

Depending on the return value of your function, you can use it to populate properties on Form UI Controls or other Form Functions.

2 Moving on

In the steps shown above we created a single function – if you create aditional classes like DemoFunctionFactory and DemoFunction and register them with an aditional <add /> element in Composite.configuration you can build a small library of Form Functions.

Form Functions are also great for providing options for items like select boxes. If your function returns a structure like Dictionary<Guid, string> you can use it in a setup like this:

```
<KeySelector Label="Category" Help="Category" OptionsKeyField="Key"
OptionsLabelField="Value"
xmlns:cms="http://www.composite.net/ns/management/bindingforms/1.0">
  <KeySelector.Selected>
    <cms:bind source="Id" />
  </KeySelector.Selected>
  <KeySelector.Options>
    <CategoryOptions xmlns="http://www.mycompany.net/ns/c1/formfunctions">
      <CategoryOptions.PageId>
        <cms:read source="PageIdForeignKey" />
      </CategoryOptions.PageId>
    </CategoryOptions>
  </KeySelector.Options>
</KeySelector>
```

This markup invokes the default selector which gets its options from a function registered as "CategoryOptions" living in the namespace http://www.mycompany.net/ns/c1/formfunctions. The function is passed data from the forms bindings collection.