

# Sitecore Forms Module Developer's Guide

# 1 Contents

1	Contents.....	2
2	Synopsis.....	3
3	Forms Setup .....	4
3.1	Form Fields .....	4
3.2	Field Definitions .....	5
3.2.1	Example.....	6
3.3	Field Styles, Attributes and Events .....	6
4	Developing Functions .....	8
4.1	Function Declaration .....	8
4.1.1	FormsFunction .....	9
4.2	Declaring and Using Parameters.....	9
4.2.1	ValueParameter .....	9
4.2.2	ControlParameter .....	9
4.2.3	ListParameter .....	10
4.2.4	SitecorePathParameter .....	10
5	Adding functions to Forms.....	12
6	Assigning default values to functions .....	13

## 2 Synopsis

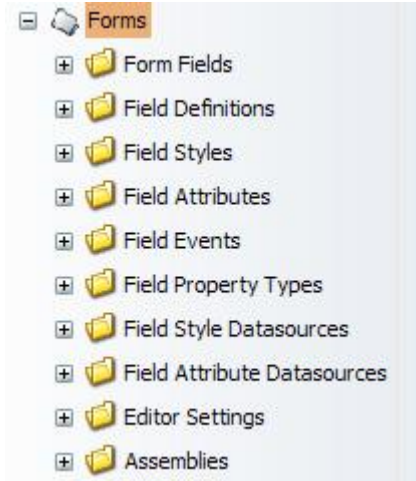
Forms are a forms editor, enabling you to create Web forms with a drag-n-drop editor. The forms you create can be fully customized to meet any requirement you might have.

You have full control over the location and design of your controls.

This manual explains how to set up the forms system, and how to develop new functions. Developing your own function only applies to the Pro version of Forms.

## 3 Forms Setup

The forms setup is located under /system/modules/forms:



**Figure 1 Forms Setup**

- Form Fields: The fields tree as displayed in the forms editor.
- Form Definitions: Definitions of available controls.
- Field Styles: Definition of available styles.
- Field Attributes: Definition of available control attributes.
- Field Events: Definition of available events.
- Field Property Types: System information. Do not change
- Field Style Datasources: Definitions of the contents of dropdown boxes in the Styles attributes.
- Field Attribute Datasources: Definitions of the contents of dropdown boxes in the Attributes,
- Editor Settings: Settings for the editor.
- Assemblies: List of all assemblies containing functions that can be hooked up to events.

### 3.1 Form Fields

This section defines the "Fields" tree as displayed in the forms editor. A field can be a control, a validator, or any other type that can be defined by HTML or .NET. In this manual, they are all named controls.

You can add custom controls to the tree by selecting "New"|"Forms.Form Field".

Field Name	Description
Identification	The name of the control
Type	Select the control type. The dropdown box reads from /system/modules/forms/field definitions. See chapter 3.2, Field Definitions for a description of how to make field definitions
Remote type	Used to hook the control into third-party modules, such as CrossFlows
Single	Check this box if you wish to make sure that there is only one control of this type on a form

Add to toolbar	Check this box if you wish to add a shortcut to the control to the forms functions toolbar
----------------	--

On the System tab, use the Icon property to define an icon for your control.

### 3.2 Field Definitions

This section defines all controls for the entire forms. It is from this list the "Type" field in "Form Fields" gets its value.

Every control has an internal and external representation. The internal representation defines how the forms editor should present the control. The external representation defines what data is written to the ascx file when the form is saved.

Usually, the internal representation is a typical HTML element definition (example: `input type="button"`), whereas the external representation is the .NET control definition (for example: `asp:Button`).

The internal/external representation method is used because of the restrictions in how a browser is able to display a form field. For example, when defining a button, you cannot draw a button resembling a button unless you use the HTML element definitions. If you used the .NET definition, the button would be drawn as a weird looking box instead.

On the "Data" tab you find the following fields:

Field Name	Description
External Element Prefix	The control namespace. Usually "asp"
External Element Name	The control name. Together with the prefix, this property defines the control name. For example <code>asp:Button</code> For custom functions, this is the name of the class.
External Element Assembly	If the external representation is a .NET control, type in the assembly name where the control is defined. For example "System.Web" For custom functions, this is the file name of the dll, but without the dll extension.
External Element URI	If the external representation is a .NET control, type in the namespace where the element is defined. For example <code>System.Web.UI.WebControls</code> . For custom functions, this is the namespace of the class.
External Element Default Attributes	Any attributes that is added to the external element by default. For example <code>myAttribute="Value"</code>
Internal Element Name	The name of the internal representation of the control. For example "input"
Internal Element Type	The type attribute name for the internal representation of the control. For example "button"
Internal Element Default Attributes	Any attributes that is added to the internal element by default. Example <code>value="Send"</code>
Internal Element Innertext	The value of the innertext HTML for the internal representation of the control.

On the "Editable Properties" tab you find the following fields:

Field Name	Description
------------	-------------

Styles	A selected list of all available styles for the external representation of the control. The styles selected must be supported by the external representation of the control. The available style are read from <code>"/System/Modules/Forms/Field Styles"</code>
Attributes	A selected list of all available attributes for the external representation of the control. The attributes selected must be supported by the external representation of the control. The available attributes are read from <code>"/System/Modules/Forms/Field Attributes"</code>
Events	A selected list of all available events for the external representation of the control. The events selected must be supported by the external representation of the control. The available events are read from <code>"/System/Modules/Forms/Field Events"</code>
Accept children	Check this box if your control is able to contain other controls. Used in the group box.

### 3.2.1 Example

If we take the definition of a button as an example, you would define the external section like this:

Prefix: asp  
Name: Button  
Assembly: System.Web  
URI: System.Web.UI.WebControls

The button would be outputted as:

```
<asp:Button runat="server" />
```

The assembly and URI are added to the top of the ascx file:

```
<%@ register TagPrefix="asp" Namespace="System.Web.UI.WebControls"
Assembly="System.Web" %>
```

The internal representation would be defined like this:

Name: input  
Type: button  
Attributes: value="send"

The button would be drawn as:

```
<input type="button" value="send"/>
```

## 3.3 Field Styles, Attributes and Events

These sections define all available field styles, attributes and events.

Field styles define the visual settings for a control.

Field Attributes define attributes that is written to the control.  
Field Events define which events can be thrown by a control.

Styles, attributes and events is attached to controls via the "Styles", "Attributes" and "Events" fields in "Field Definitions". Be sure that the control supports the style.

Field Name	Description
Description	The name of the style
Type	The style type: Dropdown: Displays a dropdown box. The output value is defined in the field datasource DropdownField: Displays a dropdown box with the names of the fields currently available on the current node. DropdownID: Displays a dropdown box, but as a contrast to the Dropdown box adds an empty selection at the top of the dropdown, and uses the Sitecore ID as value. Text: Displays a textbox.
Datasource	Used when type is Dropdown. Defines where the values in the drop down box are taken from
Default value	Defines the default value of the style/attribute

## 4 Developing Functions

The Forms module has an open API where you can develop your own functions. Functions are attached to events via the Functions Manager.

All standard functions are supplied as open source. Use these functions as a reference when developing new functions.

When you develop your own functions, you collect them in assemblies. You register your assembly by adding it to System/Modules/Forms/Assemblies. See also chapter 5, Adding functions to Forms.

Here is an example of a class with a function:

```
public class MyFunctions
{
    [FormsFunction("The function description")]
    [ValueParameter("Control01",
        Description = "A value parameter",
        Default="Default value")]
    [ControlParameter("Control02",
        Description="A control parameter")]
    static public void MyFunction(BaseForm o, Control Sender,
        BasePageEventArgs args)
    {
        string myValue = args.Parameters["Control01"];
        string myControlID = args.Parameters["Control02"];

        if (myControlID == "")
            throw new Exception("MyFunction - Control02: parameters must not be empty");

        Control myControl = o.FindControl(controlID);

        if (myControl == null)
            throw new Exception("MyFunction - Control02: could not find control");

        // Do something with the value and the control
    }
}
```

You must include Forms.Common, Forms.Engine and System.Web in the list of assemblies. The Custom attributes are defined in the Sitecore.Modules.Forms namespace.

### 4.1 Function Declaration

The functions you develop **must** have the following specification:

```
static public void MyFunction(BaseForm o,
    Control Sender,
    BasePageEventArgs args)
```

The function must be declared as static public void, and have the BaseForm, Control and BasePageEventArgs classes as parameters. It is only the function name that you can alter.



### 4.1.1 FormsFunction

This custom attribute define the description for your function:

```
[FormsFunction("This is a description of your function")]
```

The minimum definition of a function is:

```
[FormsFunction("This is a description of your function")]  
static public void MyFunction(BaseForm o,  
                             Control Sender,  
                             BasePageEventArgs args)
```

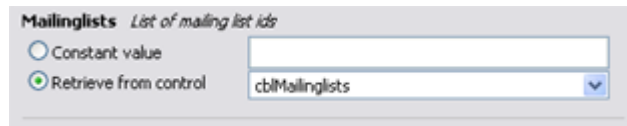
This defines a function with no parameters.

## 4.2 Declaring and Using Parameters

Declaring the parameters is done with custom attributes, FormsFunction, ValueParameter, ControlParameter, ListParameter and SitecorePathParameter.

### 4.2.1 ValueParameter

This custom attribute is assigned to either a control or a constant value.



**Figure 2 Sample ValueParameter**

The value returned is always a string.

If it is assigned to an edit box, the text of the box is returned. If assigned to a checkbox, the text 0 or 1 is returned. If it is assigned to a list box, values are returned as pipe separated values.

Definition of a ValueParameter is:

```
[ValueParameter("Parameter Name", Description="Description of value",  
Default="Default value")]
```

Using the value in the code:

```
string myargument = args.Parameters["Parameter Name"];
```

### 4.2.2 ControlParameter

This custom attribute is assigned to a control on the form and returns the id of the control. This is used if you wish to query any of the attributes on a control.



**Figure 3 Sample ControlParameter**

Definition of ControlParameter is:

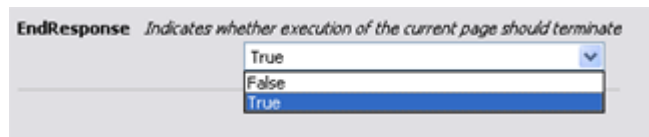
```
[ControlParameter("Parameter Name", Description="Description of value")]
```

Using the value in the code:

```
String controlID = args.Parameters["Parameter Name"];  
Control control = o.FindControl(controlID);
```

### 4.2.3 ListParameter

This custom attribute equals the ValueParameter, but has a defined list of available values attached.



**Figure 4 Sample ListParameter**

Definition of ListParameter is:

```
[ListParameter( "Parameter Name","List|of|values", Description = "Description of value")]
```

The list of values (the second parameter) is a pipe (|) separated list of available values. The first value in the list is the default value.

```
[ListParameter( "EndResponse","False|True", Description="Indicates whether execution of the current page should terminate.")]
```

The EndResponse has 2 values, False and True.

If you wish to have an empty value, start the string with a pipe:

```
[ListParameter( "Param1","|1|2|3", Description="")]
```

The values will be <empty>, 1, 2 and 3.

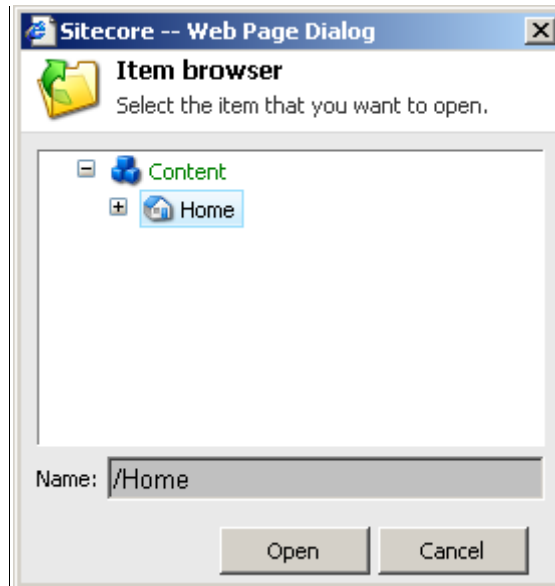
### 4.2.4 SitecorePathParameter

This custom attribute is used to find a Sitecore node:



**Figure 5 Sample SitecoreParameter**

Users can press the [...] button to open the Sitecore item browser:



**Figure 6 Sitecore Item Browser**

The resulting value is a sitecore path, not a GUID.

Definition of SitecorePathParameter is:

```
[SitecorePathParameter ( "Parameter Name", Description = "description",  
Default="sitecore node")]
```

For example:

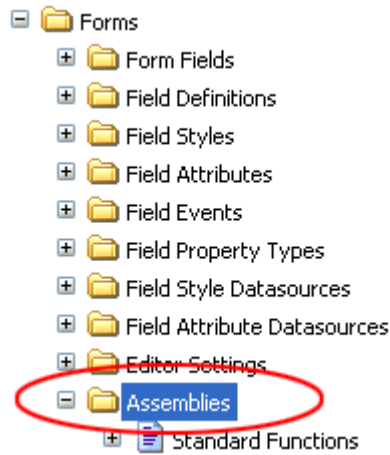
```
[SitecorePathParameter ( "SitecorePath", Description = "Sitecore path to retrieve the  
value from", Default="/sitecore/content/home")]
```

Using the value in the code:

```
string sitecorePath = args.Parameters["SitecorePath"];
```

## 5 Adding functions to Forms

To adding your functions to Forms, add a new node under /system/modules/forms/assemblies:



**Figure 7 Assemblies**

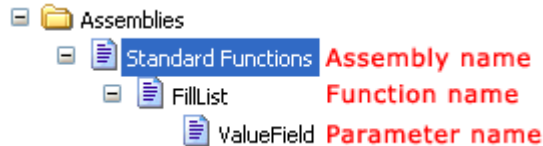
Enter the name in the name field, and the assembly name in the Assembly field.

## 6 Assigning default values to functions

When you have created an assembly, you can override the default values to functions.

These default values resemble the "branch template" term in Sitecore. It is used to override any default values given by the developer.

Under your assembly, add a node with the name of the function for which you will assign a default value. Under the function node, add a node with the name of the parameter:



**Figure 8 Default values**

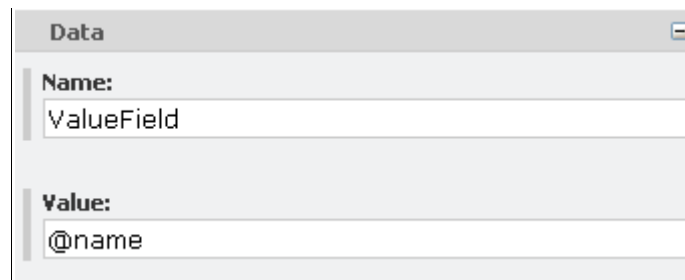
The first node describes the assembly. Write the assembly name in the field "Assembly", for example "Forms.Engine".

The second node describes the function. Write the function name in "Name", class name in "Class" and the namespace in "Namespace". Remember that this is case sensitive.

If the function MyFunction relies in the class Sitecore.Modules.Forms.MyFunctions, write "MyFunction" in "Name", "MyFunctions" in "Class" and "Sitecore.Modules.Forms" in "Namespace".

The third node describes the parameter and the default value.

The ValueField parameter has in the example been given the default value @name:



**Figure 9 Default value**