

# Axendo Form Designer

[www.axendo.nl/axendoformdesigner](http://www.axendo.nl/axendoformdesigner)

**Remarks:** Documentation version 1, October 30<sup>th</sup>, 2009

**Creator:** Ron Brouwer, Axendo

**Copyright:** Axendo, All Rights Reserved

# Table of contents

<b>1</b>	<b>FEATURE SUMMARY</b>	<b>4</b>
<b>2</b>	<b>INSTALLATION</b>	<b>5</b>
2.1	REQUIREMENTS	5
2.2	INSTALLING THE FORM DESIGNER PACKAGE	5
2.3	CONFIGURING THE FORM DESIGNER PACKAGE	5
<b>3</b>	<b>CONFIGURATION</b>	<b>6</b>
3.1	TINYMCE	6
3.2	SUBMITHANDLERS	6
3.3	VALIDATORS	6
3.4	CAPTCHASETTINGS	6
3.5	INFORMATIONICONPATH	6
3.6	DEFAULTCONFIGURATION	6
<b>4</b>	<b>IMPLEMENTING FORMS</b>	<b>7</b>
4.1	CREATING A TEMPLATE SPECIFICALLY FOR THE DOCUMENTTYPE	7
4.2	CREATING A FORM FOR SEVERAL TEMPLATES	8
4.3	USING THE FORM DESIGNER MACRO IN YOUR CONTENT	8
<b>5</b>	<b>CREATE FORMS</b>	<b>9</b>
5.1	GENERAL	9
5.2	CAPTCHA	10
5.3	CHECKBOXLIST	10
5.4	CONTENTRESPONSE (ADVANCED) (DEVELOPERS ONLY)	11
5.5	DROPDOWNLIST	12
5.6	FILEUPLOADER	12
5.7	PLAINTEXT	13
5.8	RADIOBUTTONLIST	14
5.9	SUBMITBUTTON	14
5.10	TEXTAREA	15
5.11	TEXTBOX	15
5.12	WYSIWYGTEXT	16
<b>6</b>	<b>MAILING</b>	<b>17</b>
<b>7</b>	<b>CREATE NODES</b>	<b>18</b>
<b>8</b>	<b>REDIRECTING TO URL</b>	<b>19</b>
<b>9</b>	<b>PLACEHOLDERS</b>	<b>20</b>
<b>10</b>	<b>USING MACROS</b>	<b>21</b>

<b>11</b>	<b>EXTENDING AXENDO FORM DESIGNER (DEVELOPERS)</b>	<b>22</b>
11.1	CREATING A SUBMITHANDLER	22
11.2	SUBMITHANDLEREVENTARGS	23
11.3	CREATING A FORMCONTROL	24
11.4	CREATING A SUBMITCONTROL	29
11.5	USING THE HELPER CLASS	29
11.6	USING THE CONFIGURATION CLASS	30
11.7	USING THE DYNAMICLISTPROVIDER	31

# 1 Feature summary

Manageable from the content section of Umbraco

Easy to use graphical interface

Integrated Javascript, HTML and WYSIWYG editor

Easy to use and customizable form validation

100% W3C compliant

Accessible front-end (WCAG)

Easy editable look and feel through CSS

Protect your forms against spam with a CAPTCHA (**C**ompletely **A**utomated **P**ublic Turing test to tell **C**omputers and **H**umans **A**part.)

Allow multiple selections on your form using a CheckBoxList

Allow single selection on your form using a DropDownList

The FileUploader makes it possible to attach files when submitting a form

Allow single selection from directly visible options using a RadioButtonList

Show multiple line input fields with TextAreas

Show single line input fields with TextBoxes

Send one or more emails with submitted information to anyone you can think of

Create nodes with submitted data anywhere you want

Redirect users to a page after submitting a form

Create your own SubmitHandlers

Create your own FormControls

Create your own SubmitControls

## 2 Installation

### 2.1 Requirements

Make sure you are running .NET 3.5 and Umbraco v4.0.1 or later.

### 2.2 Installing the Form Designer package

Go to the developer section of umbraco and then go to packages. Click on “Install local package” and select the Form Designer zip-file. Make sure you check the “I understand the security risks associated with installing a local package” box and press “Load Package”. In the next screen check the “Accept license” box and press “Install Package”.

### 2.3 Configuring the Form Designer package

Go to your umbraco folder and open config/AxendoLicenses.config

Enter your Form Designer key as followed:

```
<Axendolicens>  
<package name="AxendoFormDesigner">  
<key>  
ReplaceThisLineWithYourKey  
</key>  
</package>  
</Axendolicens>
```

You can find a 30-day trial key at: <http://www.axendo.nl/axendoformdesigner-installation-guide>

Save the config file and restart/reset your application.

You are now ready to use Form Designer.

## 3 Configuration

In the AxendoFormDesigner config you can add or change settings for various Axendo Form Designer features.

### 3.1 TinyMCE

This area is used for custom config settings that should be added during TinyMCE initialization. For a complete list of configuration options you can go to <http://wiki.moxiecode.com/index.php/TinyMCE:Configuration> . This customization can be applied to WYSIWYGText, Mailer and ContentResponse.

### 3.2 SubmitHandlers

Here you can add or change the SubmitHandlers. SubmitHandlers are invoked in the order that they are displayed. The Redirector SubmitHandler should always be listed last because it will cause all SubmitHandlers listed below it to be ignored.

### 3.3 Validators

Here you can add or change the validators that are used in the form designer.

### 3.4 CaptchaSettings

Here you can change the settings for the captcha display.

### 3.5 InformationIconPath

This is the path where the information icon is found.

### 3.6 DefaultConfiguration

This area is used for defaultsettings for form elements and form controls. Every form element can be set individually. The form controls default values are set by XML.

## 4 Implementing forms

After installing Form Designer, an example documenttype is created. This documenttype includes the datatype “Axendo Form Designer”. Using this datatype makes the documenttype usable as a form. There are several ways of implementing this documenttype in your website.

You can also make your own custom documenttype for your forms with the following datatypes:

**Axendo Form Designer** – This datatype makes it possible to assemble a form.

**Axendo Form Mailer** – This datatype allows you to send emails to various addresses.

**Axendo Form Nodecreator** – This datatype allows the creation of nodes when submitting forms.

**Axendo Form Redirector** – This datatype lets you redirect users after submitting a form.

**Axendo Form Placeholders** – A list of placeholders that are active in the form.

### 4.1 Creating a template specifically for the documenttype

You can create a template specifically for the documenttype which includes the Axendo Form Designer datatype. If you want to do this you will have to implement the Axendo Form Designer macro in your template. **This macro should always be placed inside the <form> element!**

The Form Designer macro is added when installing the package and looks as following:

```
<umbraco:Macro RenderContent="" Alias="AxendoFormDesigner"
runat="server"></umbraco:Macro>
```

If you do not select a specific form when you implement the macro, then *RenderContent* will be empty by default. If *RenderContent* is empty, it will always look at the current page (read: the documenttype of the page you’re using the template for) for an Axendo Form Designer datatype.

These are the steps to make the form appear on your site with this technique:

1. Create a template which includes the Form Designer macro.
2. Allow your created template to be used for the Form Designer documenttype.
3. Create a node where you want the form to appear with the Form Designer documenttype.
4. Select your created template for the node.
5. Start building your form with the Form tab. (See chapter 5.)

## 4.2 Creating a form for several templates

If you want to create a form that will be used on several different templates you can copy all the templates and create them again with the Form Designer macro in it. But a better approach would be to simply implement the macro inside the existing template, but this time select the node of the Form Designer. This will make the macro look as following:

```
<umbraco:Macro RenderContent="SelectedFormDesignerNodeID" Alias="AxendoFormDesigner"
runat="server"></umbraco:Macro>
```

What happens now is that instead of making a node with a newly created template specifically for the form, you can use the existing template and refer to the node of the Form Designer. This way, that node can be used for several templates.

These are the steps to make the form appear on your site with this technique:

1. Create a node with the Form Designer documenttype.
2. Implement the macro on the template you want it to show and select the content node of the Form Designer.
3. Start building your form with the Form tab of your Form Designer node. (See chapter 5.)

## 4.3 Using the Form Designer macro in your content

The macro can also be implemented in the content of your page. This way, you will not have to alter any templates, but just add the Form Designer macro to the content where you want it to appear on your site. Make sure you select the Form Designer node when you do this because the page itself has no Form Designer datatype and therefore will not show anything.

After implementation the macro should look as following:

```
<umbraco:Macro RenderContent="SelectedFormDesignerNodeID" Alias="AxendoFormDesigner"
runat="server"></umbraco:Macro>
```

These are the steps to make the form appear on your site with this technique:

1. Create a node with the Form Designer documenttype.
2. Implement the macro in the content where you want it to show and select the content node of the Form Designer.
3. Start building your form with the Form tab of your Form Designer node. (See chapter 5.)

## 5 Create forms

The form tab is the first tab of the example documenttype where you add the elements to your form. Here is a list of the different kinds of form elements.

### 5.1 General

The general options are visible on all the different form controls.

**General:**

Label

Information

Show label

Position **Top**   
**Left**

New line

Indent

Label:	The label of the form element.
Information:	Information that is displayed when hovering over the information icon which will only be shown if this field is filled out.
Show label:	Check if you want the label to be displayed on your form.
Position:	Select the position of the label in relevance to the form element field.
New line:	Check if you want the form element to start on a new line.
Indent:	Check if you want the form element field to indent.

## 5.2 Captcha

**Captcha:**

Width

Settings

Validation Message

Width: The width of the captcha element field.  
Settings: Select your captcha settings here.  
Validation Message: The message that is displayed when the captcha does not validate.

## 5.3 CheckBoxList

**CheckBoxList:**

Listtype

Choices  
(One choice per line)

Minimum choices

Minimum Message

Maximum choices

Maximum Message

Repeat colums

Repeat direction  Horizontal  Vertical

Listtype: The type of checkboxlist.  
Static: A custom list  
Content: A list of content of your website under a selected node. This can be filtered by document type and you can select the property of the nodes that will be visible in the list.  
Languages: A list of languages.  
Countries: A list of countries.  
Choices: Enter the labels of the checkboxes.  
Minimum Choices: Set the minimum amount of boxes that need to be checked when filling out the form.

- Minimum Message: The message that will be displayed when too few boxes have been checked when filling out the form.
- Maximum Choices: Set the maximum amount of boxes that are allowed to be checked when filling out the form.
- Maximum Message: The message that will be displayed when too many boxes have been checked when filling out the form.
- Repeat Columns: The amount of columns you want your choices to be listed in on the form.
- Repeat Direction: The direction you want your choices to be listed in.

#### 5.4 ContentResponse (Advanced) (Developers only)

**ContentResponse (Advanced):**

Parameter	
Text	

HTML | **B** | *I* | U | | | | | | | | | Format ▾

- Parameter: The name of the querystring key.
- Text: The text that will be shown as the content response.

For example:

Let's assume the label of the ContentResponse would be "Product" and the URL would contain "/Product/?Productid=10". We set the parameter to "Productid" and the text to "Question about [%ProductTitle%]". The parameter would get the Productid (10 in this case) which refers to the product in question. Now the placeholder in the text element knows from which node it should get the ProductTitle.

In this case the placeholder of the documenttype property of the ContentResponse looks as following:

[%Product:ProductTitle%]

## 5.5 DropDownList

<b>DropDownList:</b>	
Listtype	<input type="text" value="Static"/>
Choices (One choice per line)	<div style="border: 1px solid black; height: 100px;"></div>
Width	<input type="text" value="200"/>
Required	<input type="checkbox"/>
Required Message	<input type="text"/>

- Listtype: The type of dropdownlist.  
Static: A custom list  
Content: A list of content of your website under a selected node. This can be filtered by document type and you can select the property of the nodes that will be visible in the list.  
Languages: A list of languages.  
Countries: A list of countries.  
Choices: The labels of the dropdownlist items.  
Width: The width of the dropdownlist.  
Required: Check if you want the dropdownlist to be required when filling out the form.  
Required Message: The message that is displayed when no selection has been made from a required dropdownlist.

## 5.6 FileUploader

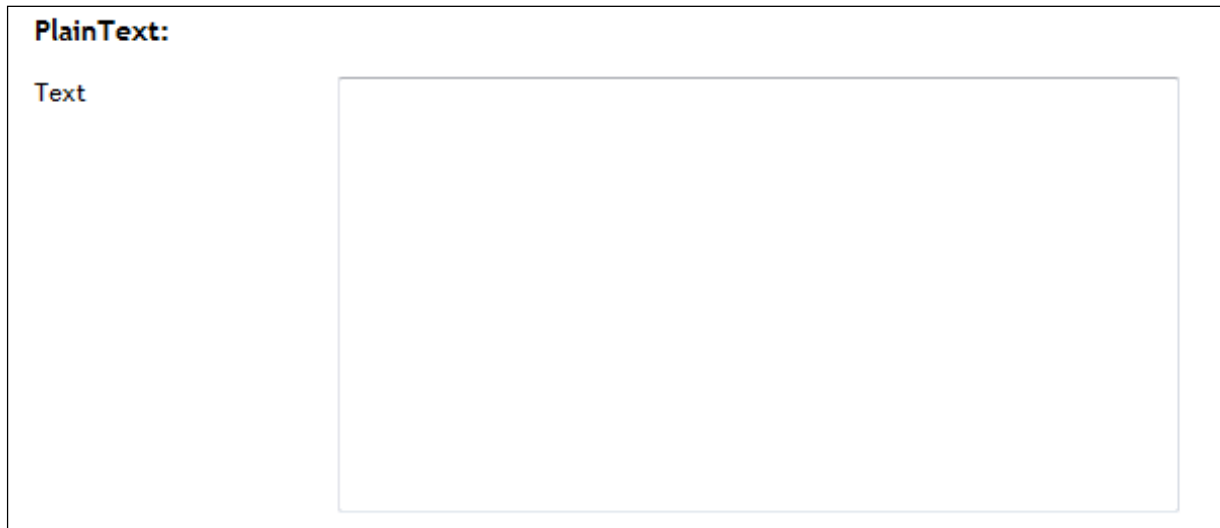
**At the moment, the FileUploader only works in the Mailer datatype.**

<b>FileUploader:</b>	
Width	<input type="text"/>
Required	<input type="checkbox"/>
Required Message	<input type="text"/>
PreferredFileName	<input type="text"/>
Accepted Mimetypes	<input type="text"/>

- Width: The width of the fileuploader field.

- Required: Check if you want the fileuploader to be required when filling out the form.
- Required Message: The message that will be displayed when no file is selected for a required fileuploader.
- Preferred File Name: The name that will be given to the uploaded file.

## 5.7 PlainText



The diagram shows a rectangular container representing a form element. Inside the container, the text "PlainText:" is positioned at the top left. Below it, the word "Text" is displayed. To the right of "Text" is a large, empty rectangular box with a thin border, representing the area where the text will be displayed.

Text: The text that will be displayed on your form in the plaintext element.

## 5.8 RadioButtonList

**RadioButtonList:**

Listtype

Choices  
(One choice per line)

Required

Required Message

Repeat colums

Repeat direction  Horizontal  Vertical

- Listtype: The type of radiobuttonlist.  
Static: A custom list  
Content: A list of content of your website under a selected node. This can be filtered by document type and you can select the property of the nodes that will be visible in the list.  
Languages: A list of languages.  
Countries: A list of countries.
- Choices: The labels of the radiobuttonlist items.
- Required: Check if you want the radiobuttonlist to be required when filling out the form.
- Required Message: The message that will be displayed when a required radiobuttonlist is not used when filling out the form.
- Repeat Columns: The amount of columns your want your choices to be listed in on the form.
- Repeat Direction: The direction you want your choices to be listed in.

## 5.9 SubmitButton

**SubmitButton:**

Width

Text

- Width: The width of the submit button.
- Text: The text that will be shown on the submit button.

## 5.10 TextArea

<b>TextArea:</b>	
Width	<input type="text" value="200"/>
Rows	<input type="text" value="8"/>
Required	<input type="checkbox"/>
Required Message	<input type="text"/>

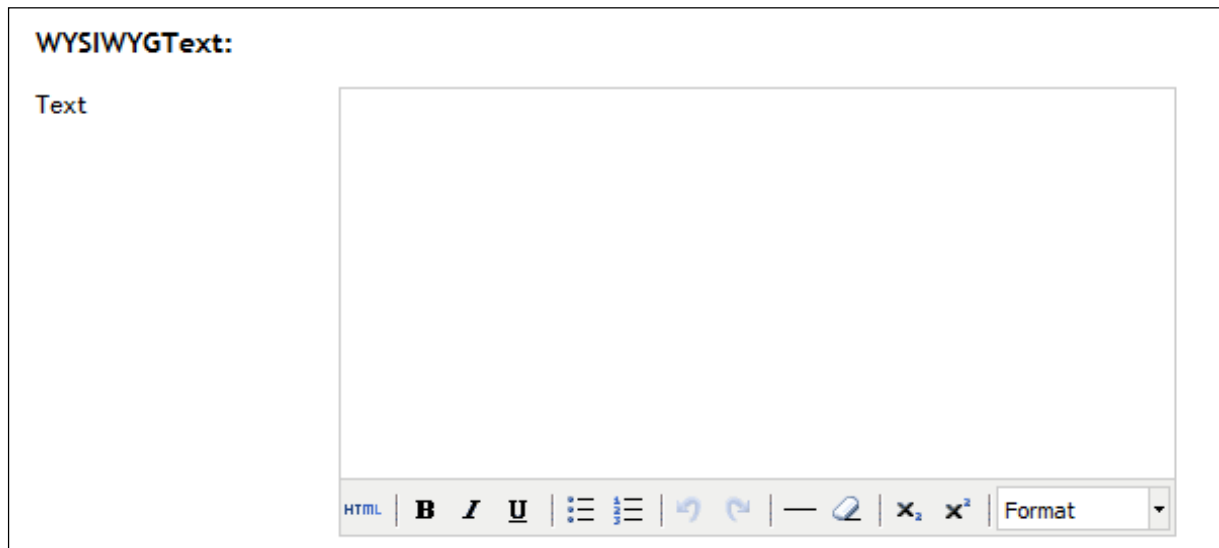
- Width: The width of the textarea.
- Rows: The amount of rows of the textarea.
- Required: Check if you want the textarea to be required when filling out the form.
- Required Message: The message that is displayed when a required textarea is empty when filling out the form.

## 5.11 TextBox

<b>TextBox:</b>	
Width	<input type="text" value="200"/>
Max length	<input type="text" value="255"/>
Required	<input type="checkbox"/>
Required Message	<input type="text"/>
Validation	<input type="text" value=""/>
Validation Message	<input type="text"/>




- Width: The width of the textbox.
- Max Length: The maximum amount of characters allowed in the textbox.
- Required: Check if you want the textbox to be required when filling out the form.
- Required Message: The message when a required textbox is empty when filling out the form.
- Validation: The type of validation that will be used for the textbox.
- Validation Message: The message that will be displayed when the textbox entry is not validated when filling out the form.

## 5.12 WYSIWYGText



Text: The text that will be displayed in the 'What You See Is What You Get' textfield.

## 6 Mailing

<b>Mail to us</b>	<b>Active</b>	<input type="checkbox"/>
	<b>From name +</b>	<input type="text"/>
	<b>From email +</b>	<input type="text"/>
	<b>Subject +</b>	<input type="text"/>
	<b>To +</b>	<input type="text"/> 
	<b>CC +</b>	<input type="text"/> 
	<b>BCC +</b>	<input type="text"/> 
	<b>Send as HTML</b>	<input type="checkbox"/>
	<b>Body template +</b>	<input type="text"/>

<b>Active:</b>	Check if you want a mail to be send to the specified email addresses after submitting the form.
<b>From name:</b>	The name of the email sender.
<b>From email:</b>	The email address of the email sender.
<b>Subject:</b>	The subject of the email.
<b>To:</b>	The email recipient(s).
<b>CC:</b>	The email CC addresses.
<b>BCC:</b>	The email BCC addresses.
<b>Send as HTML:</b>	Allows the use of HTML in the email.
<b>Body template:</b>	The content of the email.

## 7 Create nodes

Active	<input type="checkbox"/>
ParentNode	Content <a href="#">Delete</a> <a href="#">Choose...</a>
DocumentType	<input type="text"/>
Publish	<input type="checkbox"/>
Nodename +	<input type="text"/>
Mapping	<input type="text"/>

- Active:** Check if you want a node to be created every time a form is submitted.
- Parent Node:** The node under which the created nodes will be placed.
- Document Type:** The document type that the created nodes will get.
- Publish:** Check if you want the created nodes to be published right after they have been created.
- Nodename:** The name the created nodes will get.
- Mapping:** List of the fields and their values the created node will have with the selected document type.

## 8 Redirecting to URL

Redirect	Active	<input type="checkbox"/>
	Content	Content <a href="#">Choose...</a>
	Or URL +	<input type="text"/> 

**Active:** Check if you want the user to be redirected after filling out the form.  
**Content:** The page the user will be redirected to after filling out the form.  
**Or URL:** The URL the user will be directed to after filling out the form. This will be ignored if a content page has been selected as well.

## 9 Placeholders

<b>Placeholders</b>	Click once on a placeholder to copy onto clipboard (IE only) or click three times and press [Ctrl-C]. These placeholders can be used in all fields labeled with a + sign.
	<code>[%Name%]</code>
	<code>[%Surname%]</code>
	<code>[%email%]</code>
	<code>[%Languages%]</code>
	<code>[%Languages:Dutch%]</code>
	<code>[%Languages:English%]</code>
	<code>[%Languages:German%]</code>
	<code>[%Languages:Spanish%]</code>

A list of placeholders that can be used to link to the value of the corresponding fields.

In the example above `[%Languages%]` will return all selected languages, while `[%Languages:Dutch%]` will return a *yes* or *no*.

## 10 Using macros

When using macros, it is important to know that macros will be rendered by using the current page when sending the information.

Macros used inside the ContentResponse (Advanced) however, are rendered by using the node of the ContentResponse.

Valid ways of implementing a macro are:

```
<umbraco_macro macroalias="Alias" runat="server"></umbraco_macro>
```

Or:

```
<?umbraco_macro macroalias="Alias" runat="server"/>
```

## 11 Extending Axendo Form Designer (Developers)

Advanced users can create their own controls and handlers to increase the functionality of the Form Designer.

### 11.1 Creating a SubmitHandler

You can find the interface for this handler at: `AxendoFormDesigner.Interfaces.ISubmitHandler`

A summary of the SubmitHandler EventArgs is found in chapter 11.3.

1. Create a new Web Application in Visual Studio.
2. Add a reference to the `AxendoFormDesigner.dll`.
3. Add a class to the project.
4. Implement the interface `AxendoFormDesigner.Interfaces.ISubmitHandler`
5. Write your code!

```
namespace YourNamespace
{
    public class YourClassName : AxendoFormDesigner.Interfaces.ISubmitHandler
    {
        #region ISubmitHandler Members

        public void OnSubmit(object sender, AxendoFormDesigner.SubmitHandlerEventArgs e)
        {
            START WRITING YOUR CODE HERE!
        }

        #endregion
    }
}
```

It is important to note that when you want to make use of a property of a ContentResponse you can do so by using the following syntax:

```
[%ContentResponseName:PropertyName%]
```

6. Build your project and place the projects .dll-file in the sites bin folder.
7. Place the .ascx-file in `/AxendoFormDesigner/SubmitHandlers`.
8. Open the `AxendoFormDesigner.config`.

9. Add a SubmitHandler line above the SubmitHandlerRedirector.
  - a. assembly = .dll-filename without the .dll-extension
  - b. type = namespace.classname

```

</Instance>
</TinyMCE>
<SubmitHandlers>
  <!--
  <SubmitHandler assembly="/bin/name-of-dll-without-dll-extension" type="Fully.Qualified.Name.Of.Type.Including.Namespace"/>
  The SubmitHandlers are invoked in the order displayed below.
  -->
  <SubmitHandler assembly="/bin/SubmitHandlerMailer" type="SubmitHandlerMailer.Mailer"/>
  <SubmitHandler assembly="/bin/SubmitHandlerNodeCreator" type="SubmitHandlerNodeCreator.NodeCreator"/>
  <SubmitHandler assembly="/bin/YourSubmitHandlerDLLName" type="YourNamespace.YourClassname"/>
  <SubmitHandler assembly="/bin/SubmitHandlerRedirector" type="SubmitHandlerRedirector.Redirector"/>
</SubmitHandlers>
<Validators>
  <!--
  <Validator Identifier="Unique number" Name="Name of validator" Expression="Expression of validator"/>

```

10. Save the AxendoFormDesigner.config

## 11.2 SubmitHandlerEventArgs

The following method returns the node on which the form is configured.

```
public XDocument CurrentNode()
```

The following method returns the properties of all controls that are currently placed on the form.

```
public List<Property> Properties()
```

The following method returns all PlaceHolders and their respective values.

```
public List<KeyValuePair<string, object>> PlaceHolders()
```

The following method returns all EventArgs corresponding to the event that's being triggered by the SubmitControl.

```
public EventArgs OriginalEventArgs()
```

## 11.3 Creating a FormControl

You can find the interface for this control at: AxendoFormDesigner.Interfaces.IFormControl

A FormControl is any type of input you can place on your form. There are several important properties of a FormControl.

```
#region IFormControl Members
/// <summary>ValidationGroup is a string in the ExampleFormControl class.
/// <para>A validationgroup is a group that will be validated together after submitting the corresponding information.</para>
/// </summary>
public string ValidationGroup { get; set; }

/// <summary>EditMode is a boolean in the ExampleFormControl class.
/// <para>This will allow the FormDesigner to switch between Editmode and Livemode.</para>
/// </summary>
public bool EditMode { get; set; }

/// <summary>Xml is an XElement in the ExampleFormControl class.
/// <para>This will allow the FormDesigner to request and save your control settings.</para>
/// </summary>
public XElement Xml { get; set; }

/// <summary>Property is a property in the ExampleFormControl class.
/// <para>This is used to return the values of your FormControl.</para>
/// </summary>
public Property Property
{
    get
    {
        return new Property("input", txtExampleFormControl.Text);
    }
}

/// <summary>Name is a string in the ExampleFormControl class.
/// <para>This is the name of your FormControl that will be shown on the selection screen.</para>
/// </summary>
public string Name
{
    get
    {
        return "ExampleFormControl";
    }
}

/// <summary>AssociatedClientIdForLabel is a string in the ExampleFormControl class.
/// <para>This is used to link the label to the appropriate FormControl.</para>
/// </summary>
public string AssociatedClientIdForLabel
{
    get
    {
        return txtExampleFormControl.ClientID;
    }
}

#endregion
```

The previous image is part of the Axendo Form Designer FormControl Demo which can be found at: <http://www.axendo.nl/media/55957/formcontroldemo.zip>

You can create your own form control with the following steps assuming you are using the demo.

1. Fill in the name you want to show up on the selection screen for your control at the Name method.

2. Give the name to the new XElement in the GetXmlData method.

```
private void GetXmlData()
{
    if (Xml != null && ViewState["FormControlXML"] == null)
    {
        ViewState["FormControlXML"] = Xml.ToString();
    }
    else
    {
        if (ViewState["FormControlXML"] == null)
        {
            //TODO: Initialize your formcontrolelement.

            Xml = new XElement("exampleformcontrol");

            //-----End-----//

            ViewState["FormControlXML"] = Xml.ToString();
        }
        else
        {
            Xml = XElement.Parse(ViewState["FormControlXML"].ToString());
        }
    }
}
```

3. Place your input controls on the EditMode placeholder for different settings and place you controls on the LiveMode placeholder so they will be shown on the form.

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="ExampleFormControl.ascx.cs" Inherits="ExampleFormControl.ExampleFo

<asp:Placeholder ID="phEditMode" runat="server" Visible="false">
    <!--This placeholder is used to show the setting during editmode.-->
    <asp:Label ID="Label1" runat="server" AssociatedControlID="txtExampleSetting" Text="ExampleSetting"></asp:Label><asp:TextBo
    <br/>
</asp:Placeholder>

<asp:Placeholder ID="phLiveMode" runat="server" Visible="false">
    <!--This placeholder is used to show your formcontrol during livemode.-->
    <asp:TextBox CssClass="AxExampleFormControl" ID="txtExampleFormControl" Text="" runat="server"></asp:TextBox>
</asp:Placeholder>
```

#### 4. Initialize your settings in the Page\_Init method.

```
protected void Page_Init(object sender, EventArgs e)
{
    if (IsPostBack && EditMode)
    {
        GetXmlData();
        Initialize();
    }
    if (!IsPostBack && EditMode)
    {
        var t = new ExampleFormControl();

        //TODO: Initialize your settings here.

        //-----For example:-----//

        //ddlValidate.Items.Add(new ListItem(""));
        //foreach (var validationElement in Configuration.GetValidationElements())
        //{
        //    ddlValidate.Items.Add(new ListItem(validationElement.GetAttributeValue("Name"), validationElement.GetAttributeValue("Value")));
        //}

        //-----End-----//
    }
}
```

#### 5. Save your settings in the Page\_Load method.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack && EditMode)
    {
        //TODO: Save your setting here.

        Xml.SetAttributeValue("examplesetting", txtExampleSetting.Text);

        //-----End-----//

        ViewState["FormControlXML"] = Xml.ToString();
    }
    else
    {
        GetXmlData();
        Initialize();
    }
}
```

#### 6. Initialize your settings in the InitializeEditMode method.

```
private void InitializeEditMode()
{
    phEditMode.Visible = true;

    //TODO: Initialize your settings.

    txtExampleSetting.Text = Xml.GetAttributeValue("examplesetting");

    //-----End-----//
}
```

7. Initialize your FormControl in the InitializeLiveMode method. Do not forget to apply the Validationgroup! (highlighted in blue)

```
private void InitializeLiveMode()
{
    phLiveMode.Visible = true;
    txtExampleFormControl.CssClass = "AxTextBox";
    txtExampleFormControl.ValidationGroup = ValidationGroup;

    //TODO: Initialize your ExampleFormControl using your settings.

    //-----For example:-----//

    //if (Xml.GetAttributeValue("width") != "")
    //{
    //    txtExampleFormControl.Width = Unit.Parse(Xml.GetAttributeValue("width"));
    //}

    //if (Xml.GetAttributeValue("maxlength") != "")
    //{
    //    txtExampleFormControl.MaxLength = Convert.ToInt32(Xml.GetAttributeValue("maxlength"));
    //}

    //if (Xml.GetAttributeValue("required") == "True")
    //{
    //    txtExampleFormControl.CssClass += " AxRequired";
    //    var message = Xml.GetAttributeValue("requiredmessage") == "" ? "*" : Xml.GetAttributeValue("requiredmessage");
    //    var val = new RequiredFieldValidator()
    //    {
    //        ValidationGroup = ValidationGroup,
    //        ForeColor = new Color(),
    //        CssClass = "AxRequiredMessage",
    //        EnableClientScript = false,
    //        ErrorMessage = message,
    //        ControlToValidate = "txtExampleFormControl"
    //    };
    //    this.Controls.Add(val);
    //}

    //if (Xml.GetAttributeValue("validation") != "")
    //{
    //    txtExampleFormControl.CssClass += " AxValidation";
    //    var message = Xml.GetAttributeValue("validationmessage") == "" ? "*" : Xml.GetAttributeValue("validationmessage");
    //    var val = new RegularExpressionValidator()
    //    {
    //        ValidationGroup = ValidationGroup,
    //        ForeColor = new Color(),
    //        CssClass = "AxValidationMessage",
    //        EnableClientScript = false,
    //        ErrorMessage = message,
    //        ControlToValidate = "txtExampleFormControl",
    //        ValidationExpression = Configuration.GetValidationExpression(Xml.GetAttributeValue("validation"))
    //    };
    //    this.Controls.Add(val);
    //}

    //-----End-----//
}
```

8. Get your ClientID in the AssociatedClientIdForLabel method.

9. Add property values in the Property method that you want your FormControl to return.

Basic example:

```
public Property Property
{
    get
    {
        return new Property("input", txtExampleFormControl.Text);
    }
}
```

Complex example:

```
public Property Property
{
    get
    {
        var property = new Property("input");
        property.Value = fuFileUploader.PostedFile;
        property.Properties.Add(new Property("PreferredFileName", Xml.GetAttributeValue("preferredfilename").Trim()));
        property.Properties.Add(new Property("OriginalFileName", Path.GetFileName(fuFileUploader.FileName)));
        return property;
    }
}
```

10. Build your project and place the projects .dll-file in the sites bin folder.
11. Place the .ascx-file in /AxendoFormDesigner/FormControls.

You can add a selectionmenu image to your FormControl by placing an imagefile with the exact same name as your FormControl in the folder /AxendoFormDesigner/FormControllImages.

If you want to use a default configuration, you can use the XML of your control in the AxendoFormDesigner.config like other existing controls.

## 11.4 Creating a SubmitControl

You can find the interface for this control at: AxendoFormDesigner.Interfaces.ISubmitControl

A SubmitControl is an addition to a FormControl. You can create your own event like the following example:

```
#region ISubmitControl Members
public event EventHandler Submit;
#endregion
```

After that you have to invoke your event.

Example:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    if (Submit != null)
        Submit.Invoke(sender, e);
}
```

## 11.5 Using the Helper class

The namespace for this class is: AxendoFormDesigner.Utils

The Helper class has some build-in methods with functionalities that make it easier to program your Handlers and Controls.

You can use the following method in the Page\_Load of your SubmitHandler configuration to update the altered Placeholder on all used locations.

```
public static string UpdatePlaceHolders(string xml)
```

The following method replaces all the used PlaceHolders for their respective values that are filled in on the form. This also includes macros, which will be replaced last. So if the macro uses any PlaceHolders, they will be filled in before rendering the macro.

```
public static XElement MergeAllPropertiesRecursive(XElement Xml,
List<KeyValuePair<string, object>> placeHolders)
```

The following method replaces all given node properties for their respective values in the XML. This also includes "@nodeName", "@niceUrl" and macros. The macros will be replaced last, so any PlaceHolders in de macro will be filled in before rendering the macro.

```
public static XElement MergeAllNodePropertiesRecursive(XElement Xml, Node
Node)
```

This method returns the dynamic list.

```
public static List<string> GetDynamicListByConfig(string config)
```

This method converts relative URLs to absolute URLs.

```
public static String ConvertRelativePathsToAbsolute(String text)
```

## 11.6 Using the Configuration class

The namespace for this class is: AxendoFormDesigner

The Configuration class is used to get information out of the AxendoFormDesigner.config file.

The following method will return all elements of the requested setting.

```
public static IEnumerable<XElement> GetCustomElements(string parentName)
```

The following method will return the element of the requested setting based on the given identifier.

```
public static XElement GetCustomElement(string parentName, string identifier)
```

This following method will return the value of the requested attribute corresponding with the setting and element identifier.

```
public static string GetCustomAttributeValue(string parentName, string identifier, string attributeName)
```

The following method will return a list of all the validation elements.

```
public static IEnumerable<XElement> GetValidationElements()
```

The following method will return a validation based on the requested identifier.

```
public static string GetValidationExpression(string identifier)
```

## 11.7 Using the DynamicListProvider

The DynamicListProvider is used to create your own listtypes for controls like CheckBoxList, RadiobuttonList, etc. To make this work you should add the following lines to your code with a few modifications.

Add the following lines to your Page\_Load method:

```
Xml.SetAttributeValue("dynamiclistconfig", DynamicListProvider.Config);

Xml.GetOrCreateElement("choices").RemoveNodes();
foreach (var choice in
DynamicListProvider.StaticChoices.Split(Environment.NewLine.ToCharArray()))
{
    if (choice.Trim() != string.Empty)
    {
        Xml.GetOrCreateElement("choices").Add(new XElement("choice",
choice.Trim()));
    }
}
```

Add the following lines to the InitializeEditMode method:

```
DynamicListProvider =
(DynamicListProvider)LoadControl("/AxendoFormDesigner/DynamicListProvider.a
scx");
DynamicListProvider.Config = Xml.GetAttributeValue("dynamiclistconfig");

phDynamicListProvider.Controls.Add(DynamicListProvider);
foreach (var choice in
Xml.GetOrCreateElement("choices").Elements("choice"))
{
    DynamicListProvider.StaticChoices += choice.Value +
Environment.NewLine;
}
```

phDynamicListProvider (bold and underlined) is a placeholder that should be added to the EditMode.

Add the following lines to the InitializeLiveMode method:

```
if (Xml.GetAttributeValue("dynamiclistconfig").StartsWith("0"))
{
    foreach (var choice in
Xml.GetOrCreateElement("choices").Elements("choice"))
    {
        YourControl.Items.Add(new ListItem(choice.Value));
    }
}
else
{
    var choices =
Helper.GetDynamicListByConfig(Xml.GetAttributeValue("dynamiclistconfig"));
    foreach (var choice in choices)
    {
        YourControl.Items.Add(new ListItem(choice));
    }
}
```

YourControl (bold and underlined) should be replaced by the name of your control.