# Creating Custom Form Controls in the CMS Console UI

2017-02-14

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

www.orckestra.com

C1 CMS

# Contents

# 1 Creating a Simple Form Control

This document will guide you through the following steps:

1. Creating an ASP.NET User Control that inherits from **UserControlBasedUiControl**.
2. Registering the file in the *Composite.config* file.
3. Editing the markup of the form that should contain your new form control and adding the markup that activates the control.

## 1.1 Creating the User Control file

1. Create an ASP.NET User Control in a folder of your choice (e.g. */Controls/FormControls/MyTextBox.ascx*).
2. In the “code behind” file (e.g. *MyTextBox.ascx.cs*) add the required namespaces: **Composite.C1Console.Forms** and **Composite.Plugins.Forms.WebChannel.UiControlFactories**, inherit your control class from the abstract class **UserControlBasedUiControl** (instead of **System.Web.UI.UserControl**) and implement this class by adding the functions **InitializeViewState** and **BindStateToControlProperties**.
3. Add a property with a getter and a setter for each property you want your form tag to have.
   For instance, if your form tag should be like this:

```
<MyTextBox MaxLength="5" Text="Some sample text" />
```

   Then you should add two properties named “**MaxLength**” and “**Text**” to your control. The properties should be public and have the attribute **FormsProperty** specified on them.

4. If a property should be able to communicate data back and forth, it should be marked with the **BindableProperty** attribute.
5. Add the desired ASP.NET controls and markup to your User Control. In the sample code below a **TextBox** control with the **ID** of *myAspNetTextBox* is assumed.
6. Add code to **InitializeViewState** that initialize your ASP.NET controls. This function is called by the system when a form is rendered for the first time. A typical action is to copy data from your properties to your ASP.NET controls.
7. Add code to **BindStateToControlProperties** that copy user input from your ASP.NET controls to the system. This function is called by the system when a form is saved. It’s only necessary to copy data to properties with the **BindableProperty** attribute specified. Otherwise, the system won’t allow saving data from properties without this attribute.

*MyTextBox.ascx*

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="MyTextBox.ascx.cs" Inherits="Controls_FormControls_MyTextBox" %>
<asp:TextBox ID="myAspNetTextBox" runat="server">
</asp:TextBox>
```

Download the source

*MyTextBox.ascx.cs*

```
using System;
using Composite.C1Console.Forms;
using Composite.Plugins.Forms.WebChannel.UiControlFactories;
public partial class Controls_FormControls_MyTextBox :
UserControlBasedUiControl
{
    [FormsProperty()]
    public int MaxLength
    {
        get;
        set;
    }
    [BindableProperty()]
    [FormsProperty()]
    public string Text
    {
        get;
        set;
    }
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    public override void BindStateToControlProperties()
    {
        this.Text = myAspNetTextBox.Text;
    }
    public override void InitializeViewState()
    {
        myAspNetTextBox.Text = this.Text;
        myAspNetTextBox.MaxLength = this.MaxLength;
    }
}
```

Download the source

### 1.1.1 Validating User Input

The Form Control can also validate the user's input before the form is saved. For example, on DateTime fields, if the value is not valid, this will prevent the form from being saved and a validation balloon will pop up on the field with the message you specify.

To add validation to the form control, use the **IsValid** and **ValidationError** properties in the **BindStateToControlProperties** method:

```
public override void BindStateToControlProperties()
{
  try
  {
    // ...
    this.IsValid = true;
  }
  catch (Exception ex)
  {
    this.IsValid = false;
    this.ValidationError = "This is a validation error";
  }
}
```

Please note that this validation feature is available out-of-the-box on Composite C1 (now C1 CMS) v.3.2 or later. On Composite C1 (now C1 CMS) v 3.0 and 3.1, for the validation, the form controls should implement the Composite.C1Console.Forms.IValidatingUiControl interface.

## 1.2 Registering a User Control based Form UI Control

When you register your own Form UI Controls, you should first come up with, and register, an XML namespace.

Basically, this can be any unique string, but often an URI is used here.

This is important to ensure that Form UI Controls from different providers are uniquely identifiable.

## 1.3 Registering a Form UI Control namespace for the first time

Registering your own namespace is only necessary the first time and is done like this (“*http://www.contoso.com/ns/Composite/Forms/1.0*” is used here – ensure that you use your own string or URI for the namespace):

Before you proceed, ensure that the file */App_Data/Composite/Composite.modified.config* does not exist.

(If the file does exist, the system has outstanding configuration changes, which must be stored before you can continue. To fix this, restart the ASP.NET application. You can do so by recycling the application pool or by making a change to the */web.config* file (edit it, add an empty line, save it).)

1. Edit the file */App_Data/Composite/Composite.config*.
2. Locate the section **<Composite.C1Console.Forms.Plugins.ProducerMediatorConfiguration>** and the **Mediators** element below. Below it, add a new element like this (remember to customize the namespace):

```
<add
type="Composite.C1Console.Forms.StandardProducerMediators.UiControlProdu
cerMediator, Composite"
name="http://www.contoso.com/ns/Composite/Forms/1.0" />
```

3. Locate the section **<Composite.C1Console.Forms.Plugins.UiControlFactoryConfiguration>** and the **Channel** element below, with the **name** property of “*AspNet.Management*”.
4. Under the **Namespaces** element, add a new **Namespace** element, with the name of your namespace, like:

```
<Namespace
name="http://www.contoso.com/ns/Composite/Forms/1.0"></Namespace>
```

## 1.4 Registering the User Control based Form UI Control

To register your ASP.NET User Controls as a Form UI control, execute the following steps:

1. Edit the file */App_Data/Composite/Composite.config*.
2. Locate the section **<Composite.C1Console.Forms.Plugins.UiControlFactoryConfiguration>** and the **Namespace** element below, named with the namespace of your organization.
3. In the **Factories** element below your organization’s **Namespace** element, add this element:

```
<Namespace name="http://www.contoso.com/ns/Composite/Forms/1.0">
  <Factories>
    <add userControlVirtualPath="~/Controls/FormControls/MyTextBox.ascx"
name="MyTextBox" cacheCompiledUserControlType="false"
type="Composite.Plugins.Forms.WebChannel.UiControlFactories.UserControlB
asedUiControlFactory, Composite" />
  </Factories>
</Namespace>
```

Please edit the value of the properties:

- **userControlVirtualPath**: it should contain the path to your User Control file
- **name**: it should contain the name the form markup tag should have

## 1.5 Adding a custom tag to a data form

1. In the Data perspective, select the data type and click **Edit Form Markup** on the toolbar.
2. Locate the spot where you wish to add your own User Control-based Form UI Control.
3. To create a new control on the form, add the following markup:

```
<MyTextBox MaxLength="5" Text="Some sample text"
xmlns="http://www.contoso.com/ns/Composite/Forms/1.0" />
```

4. Alternatively, to bind the **Text** property to the existing data field "*MyStringField*", add the following markup:

```
<MyTextBox MaxLentgh="5"
xmlns="http://www.contoso.com/ns/Composite/Forms/1.0">
  <MyTextBox.Text>
    <bind source="MyStringField"
xmlns="http://www.composite.net/ns/management/bindingforms/1.0" />
  </MyTextBox.Text>
</MyTextBox>
```

5. Save the form markup.

# 2 Creating a Complex Form Control

The following example will show you a Form Control with an auto-updating UI, the UI that changes when the user changes some selection in it. It is a complex control (“Customer”) consisting of two drop down lists. Selecting an item in one dropdown (“Person” or “Organization”), changes the list of items in the other one (listing people or organizations respectively).

It also shows how a Form Control can bind to multiple fields of a data type. The data type has two fields of the data reference type. Each field references a specific data type (“Person” and “Organization”). The Form Control replaces two widgets assigned to those two fields by default (**KeySelector**).

Please note that for the simplicity of the example, the data sources for both fields are generated on-the-fly in the code behind the control but can be easily replaced with actual items from the corresponding data types.

The steps of adding and using this Form Control are the same as in the previous example. You should:

1. Create your control (see the examples below)
2. Register the control in Composite.config within the namespace you use for your Form Controls
3. Add the control to the data type’s form markup

## 2.1 Creating a Control

*MyControl.ascx*

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="MyControl.ascx.cs" Inherits="Controls_MyControl" %>
<%@ Register TagPrefix="aspui"
Namespace="Composite.Core.WebClient.UiControlLib" Assembly="Composite"  %>
<!-- selecting a type - changes to this will update the list of items -->
<aspui:Selector ID="typeSelector" runat="server" AutoPostBack="true">
  <asp:ListItem Text="Person" Value="Person"></asp:ListItem>
  <asp:ListItem Text="Organization" Value="Organization"></asp:ListItem>
</aspui:Selector>
<!-- the selector holding items of the selected type -->
<aspui:Selector ID="itemSelector" DataTextField="value"
DataValueField="Key" runat="server">
</aspui:Selector>
```

Download the source

In the above example, the C1 CMS’s **Selector** control is used for the drop down lists.

*MyControl.ascx.cs*

```
using System;
using System.Collections.Generic;
using Composite.C1Console.Forms;
using Composite.Plugins.Forms.WebChannel.UiControlFactories;
public partial class Controls_MyControl : UserControlBasedUiControl,
IValidatingUiControl
{
        #region Dummy Data - this could easily be dynamically loaded from
the data layer
        public Dictionary<Guid, string> PersonSource = new Dictionary<Guid,
string>()
        {
                {Guid.Empty, "select a person ..."},
                {Guid.Parse("9bdb7b6d-4e4f-41c9-bcc3-95b20bd8d791"), "Person
1"},
                {Guid.Parse("9bdb7b6d-4e4f-41c9-bcc3-95b20bd8d792"), "Person
2"},
                {Guid.Parse("9bdb7b6d-4e4f-41c9-bcc3-95b20bd8d793"), "Person
3"},
                {Guid.Parse("9bdb7b6d-4e4f-41c9-bcc3-95b20bd8d794"), "Person
4"},
                {Guid.Parse("9bdb7b6d-4e4f-41c9-bcc3-95b20bd8d795"), "Person
5"}
        };
        public Dictionary<Guid, string> OrganizationSource = new
Dictionary<Guid, string>()
        {
                {Guid.Empty, "select an organization ..."},
                {Guid.Parse("4bdb7b6d-4e4f-41c9-bcc3-95b20bd8d791"),
"Organization 1"},
                {Guid.Parse("4bdb7b6d-4e4f-41c9-bcc3-95b20bd8d792"),
"Organization 2"},
                {Guid.Parse("4bdb7b6d-4e4f-41c9-bcc3-95b20bd8d793"),
"Organization 3"},
                {Guid.Parse("4bdb7b6d-4e4f-41c9-bcc3-95b20bd8d794"),
"Organization 4"},
                {Guid.Parse("4bdb7b6d-4e4f-41c9-bcc3-95b20bd8d795"),
"Organization 5"}
        };
        #endregion
        #region Form Properties
        [FormsProperty()]
        [BindableProperty()]
        public Nullable<Guid> PersonGuid
        {
                get;
                set;
        }
        [FormsProperty()]
        [BindableProperty()]
        public Nullable<Guid> OrganizationGuid
        {
                get;
                set;
        }
        #endregion
        protected void Page_Load(object sender, EventArgs e)
        {
          // attach event to the typeSelector
                typeSelector.SelectedIndexChanged += new
EventHandler(typeSelector_SelectedIndexChanged);
        }
        #region UserControlBasedUiControl
     // this is called when the control is initialized - we should grab
values from our 'BindableProperty' fields above and show it on our ASP.NET
controls.
        public override void BindStateToControlProperties()
        {
                Guid selectedValue = Guid.Parse(itemSelector.SelectedValue);
                if (selectedValue != Guid.Empty)
                {
                        switch (typeSelector.SelectedValue)
                        {
                                case "Person": this.PersonGuid = selectedValue;
this.OrganizationGuid = null; break;
                                case "Organization": this.OrganizationGuid =
selectedValue; this.PersonGuid = null; break;
                        }
                }
                else
                {
                        this.PersonGuid = this.OrganizationGuid = null;
                }
        }
        public override void InitializeViewState()
        {
                typeSelector.SelectedValue = this.PersonGuid != null ?
"Person" : "Organization";
                FillDropDown();
                itemSelector.SelectedValue = this.PersonGuid != null ?
this.PersonGuid.ToString() : this.OrganizationGuid.ToString();
        }
        #endregion
        #region IValidatingUiControl
    // If any selections are invalid, we should return false here. We get
called for each field we bind to.
        public bool IsValid
        {
                get {
              return (this.PersonGuid != null || this.OrganizationGuid !=
null);
          }
        }
        public string ValidationError
        {
                get
                {
                        return "Missing value.\n";
                }
        }
        #endregion
        #region Private methods
        void typeSelector_SelectedIndexChanged(object sender, EventArgs e)
        {
                FillDropDown();
        }
        private void FillDropDown()
        {
                switch (typeSelector.SelectedValue)
                {
                        case "Person": itemSelector.DataSource = PersonSource;
itemSelector.DataBind(); break;
                        case "Organization": itemSelector.DataSource =
OrganizationSource; itemSelector.DataBind(); break;
                }
        }
        #endregion
}
```

Download the source

In the above example, dummy data sources are built in the code and used to fill the drop down lists with items. In an actual situation, you will get items from the corresponding data types in C1 CMS.

Please note that on Composite C1 (now C1 CMS) version 3.2 or later, you don't need to implement the IValidatingUiControl interface for user input validation and can use the validation features out-of-the-box.

## 2.2 Registering the control

In Composite.config, you should add register the control ("MyControl") under **Composite.C1Console.Forms.Plugins.UiControlFactoryConfiguration/Channels/Channel[@name="AspNet.Management"]/Namespaces/Namespace[@name="{your custom namespace for form controls}"]/Factories**:

```
<Namespace name="http://www.contoso.com/ns/Composite/Forms/1.0">
  <Factories>
    <add userControlVirtualPath="~/Controls/FormControls/MyControl.ascx"
name="MyControl" cacheCompiledUserControlType="false"
type="Composite.Plugins.Forms.WebChannel.UiControlFactories.UserControlBase
dUiControlFactory, Composite" />
  </Factories>
</Namespace>
```

Make sure you register it under the correct namespace, with the correct name and the correct path to the control.

If you have not registered a namespace for your form controls yet, add it in Composite.config under **Composite.C1Console.Forms.Plugins.ProducerMediatorConfiguration/Mediators**:

```
<add
type="Composite.C1Console.Forms.StandardProducerMediators.UiControlProducer
Mediator, Composite" name="http://www.contoso.com/ns/Composite/Forms/1.0"
/>
```

Please note that you need to register a control very time you create a new one, while registering a namespace for your controls is one-time operation provided that all the controls belong to the same namespace. You will need to register a namespace again only if you need another one.

## 2.3 Using the control on a form

In the data type's form markup, you should replace the widgets used for the respective data reference fields ("Person" and "Organization" in the example) with your form control ("MyControl").

```
<cms:formdefinition
xmlns:cms="http://www.composite.net/ns/management/bindingforms/1.0"
xmlns="http://www.composite.net/ns/management/bindingforms/std.ui.controls.
lib/1.0"
xmlns:ff="http://www.composite.net/ns/management/bindingforms/std.function.
lib/1.0" xmlns:f="http://www.composite.net/ns/function/1.0">
       <cms:bindings>
              <cms:binding name="Id" type="System.Guid" optional="true" />
              <cms:binding name="Title" type="System.String" optional="true"
/>
              <cms:binding name="Person"
type="System.Nullable`1[System.Guid]" optional="true" />
              <cms:binding name="Organization"
type="System.Nullable`1[System.Guid]" optional="true" />
       </cms:bindings>
       <cms:layout>
              <cms:layout.label>
                     <cms:read source="Title" />
              </cms:layout.label>
              <FieldGroup>
                     <TextBox Label="Title" Help="" SpellCheck="true">
                            <TextBox.Text>
                                   <cms:bind source="Title" />
                            </TextBox.Text>
                     </TextBox>
                     <MyControl Label="Customer"
xmlns="http://www.contoso.com/ns/Composite/Forms/1.0">
                            <MyControl.PersonGuid>
                                   <cms:bind source="Person" />
                            </MyControl.PersonGuid>
                            <MyControl.OrganizationGuid>
                                   <cms:bind source="Organization" />
                            </MyControl.OrganizationGuid>
                     </MyControl>
              </FieldGroup>
       </cms:layout>
</cms:formdefinition>
```