Web Forms for Marketers 2.5 for Sitecore 7.5 or later

Reference Guide

A Reference Book for Administrators and Developers

Table of Contents

Chapter 1

Introduction

This document is designed for Sitecore administrators and contains information about how to set up the module. For more detailed end users instructions, see the Web Form User Guide.

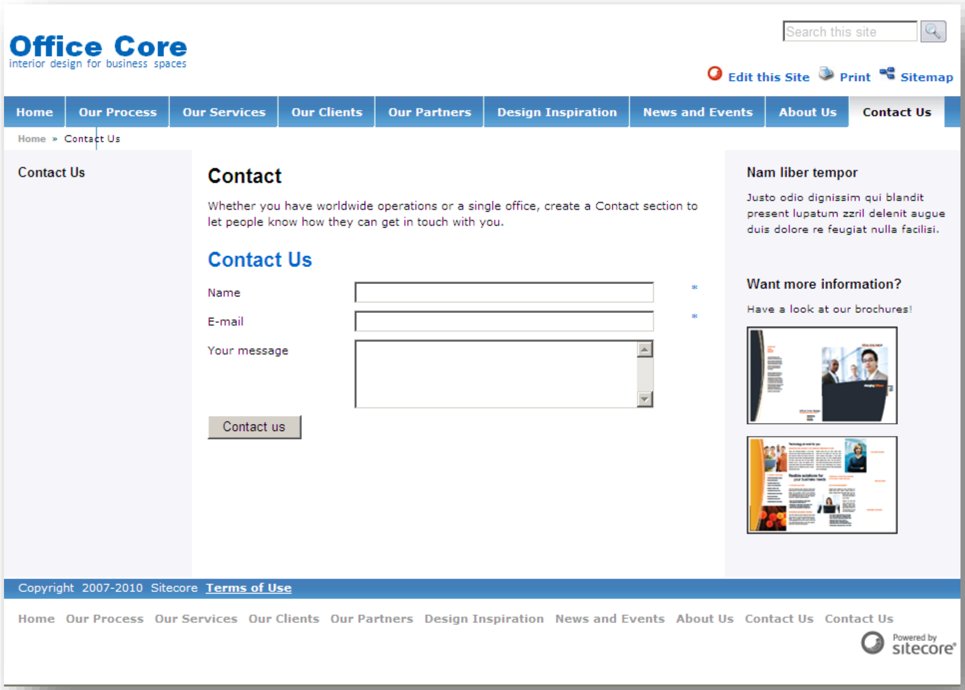
Just as most forms on a Web site are simple and straightforward, the process of creating and managing them should also be simple and straightforward.

The Web Forms for Marketers module is designed to let you create simple forms in a blink of an eye and in a user-friendly manner. The forms that you can create with this module are WCAG 2.0 and XHTML 1.0 compliant. The Web Forms for Marketers module provides users with Web analytics and reporting capabilities. It also records and reports all the information that Web site visitors enter in forms regardless of whether they successfully submit the form or not. Web Forms for Marketers 2.5 is fully integrated with the Sitecore DMS.

The module can be configured so that forms have only a few adjustable parameters thereby making the user interface as simple as possible. The basic options cover the needs of the average content editor — creating basic input fields, such as, text boxes and check boxes, creating basic actions, such as, save to a database, send an e-mail, and creating basic validators, such as, RequiredField validator, Email address validator, and so on.

You can also develop more complex forms that are based on a form generated by this module. For example, you can convert a form into a sublayout (`.ascx` control), and then edit this sublayout with development tools like Visual Studio.

Here is an example of a form created with this module:



Contact

Whether you have worldwide operations or a single office, create a Contact section to let people know how they can get in touch with you.

Contact Us

Name *

E-mail *

Your message

Contact us

Nam liber tempor

Justo odio dignissim qui blandit present lupatum zzril delenit augue duis dolore re feugiat nulla facilisi.

Want more information?

Have a look at our brochures!

Copyright 2007-2010 Sitecore Terms of Use
Home Our Process Our Services Our Clients Our Partners Design Inspiration News and Events About Us Contact Us Contact Us
Powered by sitecore

Chapter 2

Using the Module

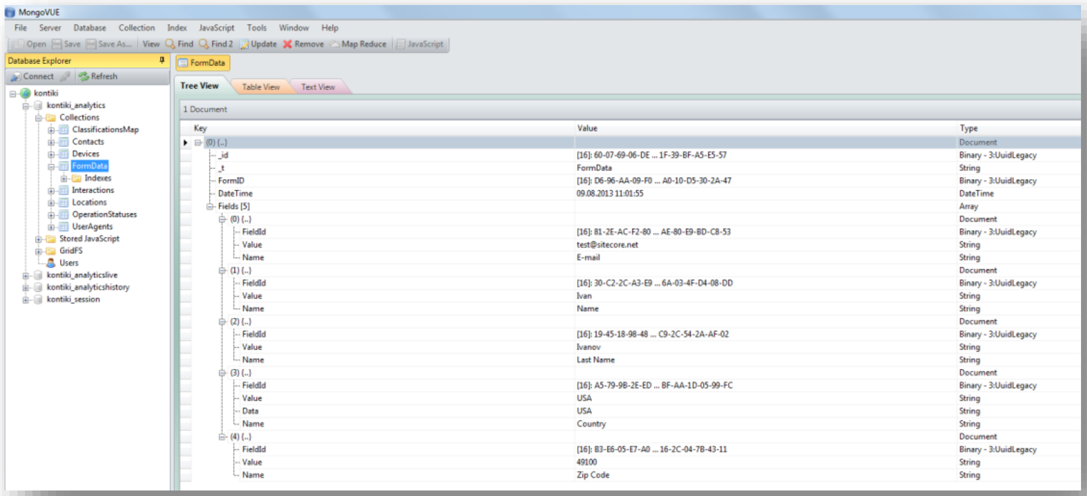
This chapter describes the functionality of the module.

This chapter contains the following sections:

2.1 Storing Web Forms

The structure of the forms is defined in the Sitecore CMS content tree.

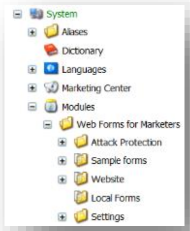
Web forms are stored in the FormData collection of the Mongo database, which is a shared resource for all Sitecore applications.



Key	Value	Type
(0) {..}		Document
_id	[16]: 60-07-69-06-DE ... 1F-39-BF-A5-E5-57	Binary - 3:UuidLegacy
_t	FormData	String
FormID	[16]: D6-96-AA-09-F0 ... A0-10-D5-30-2A-47	Binary - 3:UuidLegacy
DateTime	09.08.2013 11:01:55	DateTime
Fields [5]		Array
(0) {..}		Document
FieldId	[16]: 81-2E-AC-F2-80 ... AE-80-E9-BD-C8-53	Binary - 3:UuidLegacy
Value	test@sitecore.net	String
Name	E-mail	String
(1) {..}		Document
FieldId	[16]: 30-C2-2C-A3-E9 ... 6A-03-4F-D4-08-DD	Binary - 3:UuidLegacy
Value	Ivan	String
Name	Name	String
(2) {..}		Document
FieldId	[16]: 19-45-18-98-48 ... C9-2C-54-2A-AF-02	Binary - 3:UuidLegacy
Value	Ivanov	String
Name	Last Name	String
(3) {..}		Document
FieldId	[16]: A5-79-9B-2E-ED ... BF-AA-1D-05-99-FC	Binary - 3:UuidLegacy
Value	USA	String
Data	USA	String
Name	Country	String
(4) {..}		Document
FieldId	[16]: B3-E6-05-E7-A0 ... 16-2C-04-7B-43-11	Binary - 3:UuidLegacy
Value	49100	String
Name	Zip Code	String

The Web Forms for Marketers module has its own configuration file called `Sitecore.Forms.config`. This file is located in the `website/app_config/include` folder.

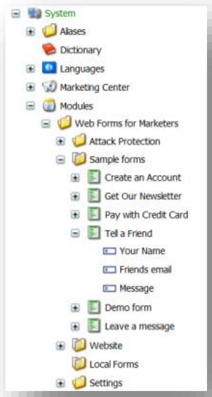
The items that make up forms are stored in appropriate folders under `/sitecore/System/Modules/Web Forms for Marketers`.



You specify where new forms are stored in the `website\app_config\include\Sitecore.Forms.config` file, in the `formsRoot` attribute of the `/sitecore/sites/site` node. If the `formsRoot` attribute is not defined for a site, new forms are created in the `/sitecore/System/Modules/Web Forms for Marketers/Local Forms` folder.

The form folders are based on the `/sitecore/Templates/Web Forms for Marketers/Forms Folder` template.

Each form is based on the `/sitecore/Templates/Web Forms for Marketers/Form` template and can contain any number of sections or fields.



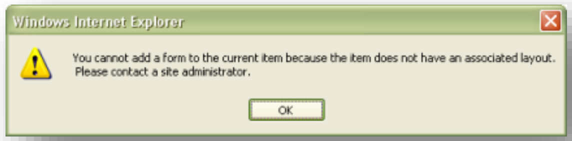
Every form contains a submit button that you can associate actions with. Actions are executed on the server.

The list of available actions is stored under the `/sitecore/System/Modules/Web Forms for Marketers/Settings/Actions` folder.

2.2 Conditions to Be Met while Creating a New Form

Users can use the **Page Editor** and the **Content Editor** to create new forms.

When a user in the **Content Editor** clicks the **Insert Form** button, the wizard first checks whether the current item has a layout assigned to it. If this condition is not met, the wizard displays the following message:



You cannot add a form to the current item because the item does not have an associated layout. Please contact a site administrator.

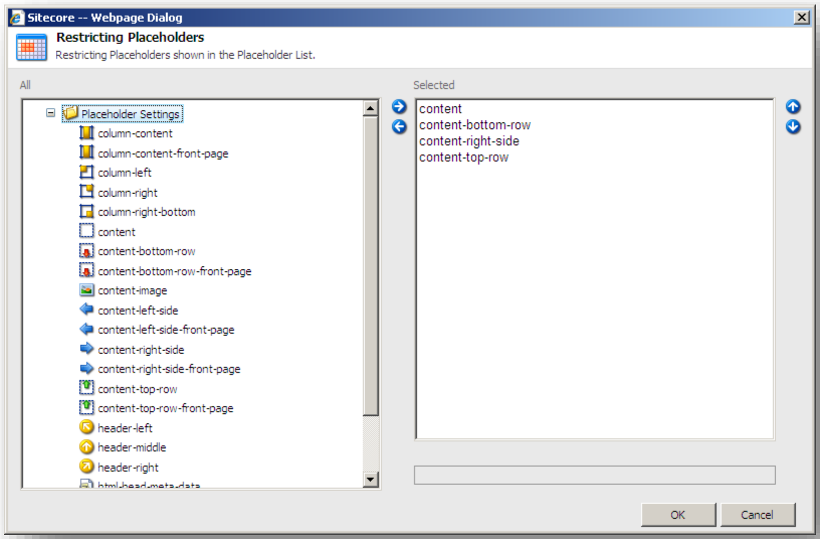
Forms renderings must be attached to a placeholder on the layout. For more information about allowed placeholders in the Web Forms for Marketers module, see the *Selecting Placeholders Shown in the Placeholder List* section.

For more information, how to insert a web form programmatically, see the section *How to Insert a Web Form on a Web Page*.

2.3 Selecting Placeholders Shown in the Placeholder List

The **Insert a New Form** wizard only allows you to add forms to placeholders that have "Placeholder Settings" items. A user who adds a new form must have write access to an item where a form is added to see placeholders in the "Placeholder list". This allows developers and website administrators to define which placeholders may contain a form.

The **Selecting Placeholders** wizard helps to select the list of placeholders shown in the **Placeholder list** of the **Insert a New Form** wizard. To run the **Selecting Placeholders** wizard click **Sitecore**, **All Applications**, **Web Forms for Marketers**, **Selecting Placeholders**. This wizard is also displayed when the Web Forms for Marketers module installation is completed.


Sitecore -- Webpage Dialog
Restricting Placeholders
Restricting Placeholders shown in the Placeholder List.
All
Selected
OK
Cancel

Placeholder Settings
- column-content
- column-content-front-page
- column-left
- column-right
- column-right-bottom
- content
- content-bottom-row
- content-bottom-row-front-page
- content-image
- content-left-side
- content-left-side-front-page
- content-right-side
- content-right-side-front-page
- content-top-row
- content-top-row-front-page
- header-left
- header-middle
- header-right
- html-head-meta-data

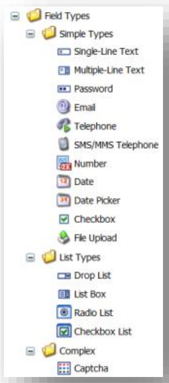
Selected:
- content
- content-bottom-row
- content-right-side
- content-top-row

The **Selected** field lists the placeholders that users can add a new form to. To add a placeholder from the **All** list to the **Selected** list, select a placeholder and click ➔. When you click **OK** all the changes are saved.

To add a placeholder to the **All** list in the **Restricting Placeholders** dialog box, you must create a new placeholder under the `Sitecore/Layout/Placeholder Settings` item.

2.4 Form Field Types

The Web Forms for Marketers module contains a number of field types that you can use to build your forms. All the form field types are stored under the `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types` item and are based on the `/sitecore/Templates/Web Forms for Marketers/Field Type` template.



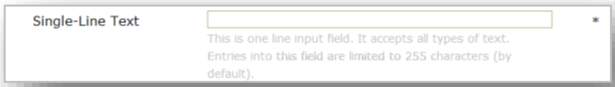
The *Field Type* template contains the following fields:

- **Assembly** — an assembly name that contains the associated class.
- **Class** — an associated class name including namespace.
- **Parameters** — save action parameters.
- **Localized Parameters** — similar to **Parameters**. The only difference is that this field is not shared so a parameter can be localized.
- **Required** — the field which defines whether the *required value* validator is applied to this field type.
- **Validation** — the list of validators that should be applied to the value entered in this field.
- **User Control** — the field defines a reference to an ASCX control.
- **Deny Tag** — the field defines whether information entered in this field type is saved to the Tag field in the *Analytics* database.

The Web Forms for Marketers module contains the following field types:

Single-Line Text

Use this field type to enter one line of text. The length of the field is limited to 255 characters by default.

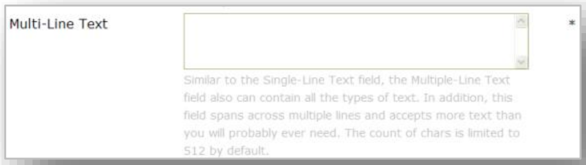

Single-Line Text

This is one line input field. It accepts all types of text. Entries into this field are limited to 255 characters (by default).

The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/Single-Line Text` item contains settings for this field type.

Multi-Line Text

Use this field type to enter multiple lines of text. The number of characters is limited to 512 by default.



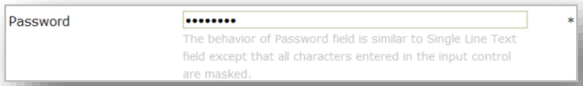
Multi-Line Text

Similar to the Single-Line Text field, the Multiple-Line Text field also can contain all the types of text. In addition, this field spans across multiple lines and accepts more text than you will probably ever need. The count of chars is limited to 512 by default.

The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/Multiple-Line Text` item contains settings for this field type.

Password

Use this field type to enter a password. All the characters you enter in a **Password** field are masked. The **Password** field is a text field.



Password

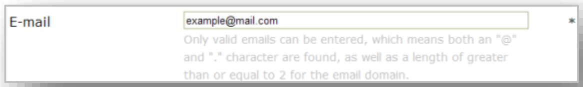
••••••••

The behavior of Password field is similar to Single Line Text field except that all characters entered in the input control are masked.

The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/Password` item contains the settings for this field type.

E-mail

Use this field type to enter e-mail addresses. The "@" and "." characters are validated, as well as the length of the e-mail server domain.



E-mail

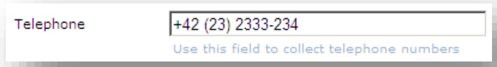
example@mail.com

Only valid emails can be entered, which means both an "@" and "." character are found, as well as a length of greater than or equal to 2 for the email domain.

The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/E-mail` item contains the settings for this field type.

Telephone

Use this field type to enter telephone numbers. This is a number field which also allows the user to enter the following characters: "+", "-", " (", ")" and spaces.



Telephone

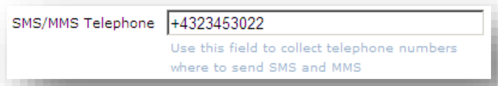
+42 (23) 2333-234

Use this field to collect telephone numbers

The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/Telephone` item contains settings for this field type.

SMS/MMS Telephone

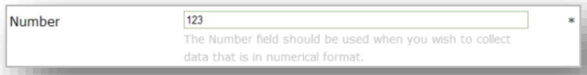
Use this field type to enter telephone numbers that you can send SMSs and MMSs to. This field allows you to enter numbers and the "+" character. The data entered can only contain the "+" character as the first symbol. This field type is used with the **Send SMS** and **Send MMS** save actions.


SMS/MMS Telephone +4323453022
Use this field to collect telephone numbers where to send SMS and MMS

The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/SMS/MMS Telephone` item contains settings for this field type.

Number

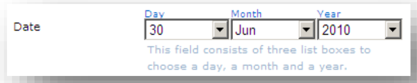
Use this field type to enter numerical data.


Number 123 *
The Number field should be used when you wish to collect data that is in numerical format.

The /`sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/Number` item contains the settings for this field type.

Date

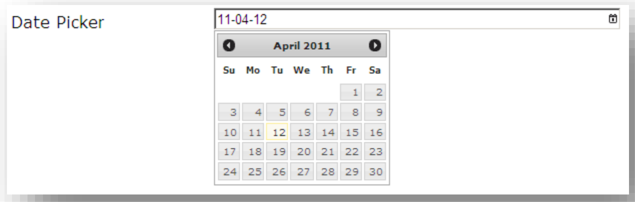
Use this field type to enter dates.


Date
Day 30 | Month Jun | Year 2010
This field consists of three list boxes to choose a day, a month and a year.

The /`sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/Date` item contains the settings for this field type.

Date Picker

Use this field type to select a date from the calendar.


Date Picker 11-04-12

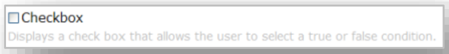
April 2011

Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

The /`sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/Date Picker` item contains the settings for this field type.

Checkbox

Use this field type to display a check box that allows you to select a true or false condition.

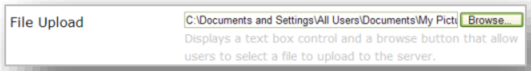

☐ Checkbox

Displays a check box that allows the user to select a true or false condition.

The /sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/Checkbox item contains the settings for this field type.

File Upload

Use this field type to display a text box and a browse button that you can use to select a file that you want to upload to the server.


File Upload

C:\Documents and Settings\All Users\Documents\My Pictu

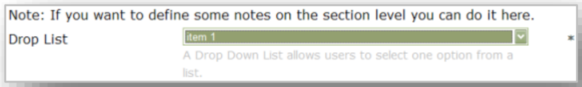
Browse...

Displays a text box control and a browse button that allow users to select a file to upload to the server.

The /sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Simple Types/File Upload item contains the settings for this field type. This field works with the master database.

Drop List

Use this field type to select an option from a list.


Note: If you want to define some notes on the section level you can do it here.

Drop List

item 1

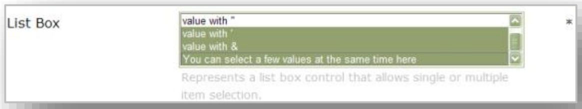
*

A Drop Down List allows users to select one option from a list.

The /sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/List Types/Drop List item contains the settings for this field type.

List Box

Use this field type to display a list box that allows you to select one or more items.


List Box

value with "

value with '

value with &

You can select a few values at the same time here

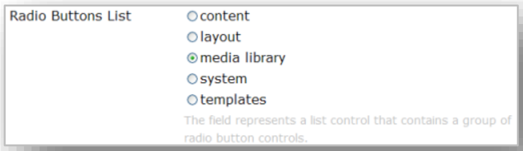
*

Represents a list box control that allows single or multiple item selection.

The /sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/List Types/List Box item contains the settings for this field type.

Radio List

Use this field type to display a group of options.



Radio Buttons List

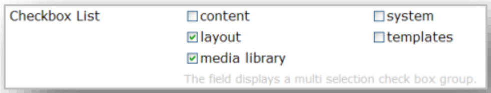
○ content
○ layout
● media library
○ system
○ templates

The field represents a list control that contains a group of radio button controls.

The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/List Types/Radio List` item contains the settings for this field type.

Checkbox List

Use this field type to display a group of check boxes. You can select one or more check boxes in the list.



Checkbox List

☐ content
☑ layout
☑ media library
☐ system
☐ templates

The field displays a multi selection check box group.

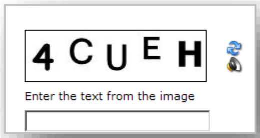
The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/List Types/Checkbox List` item contains the settings for this field type.

Section

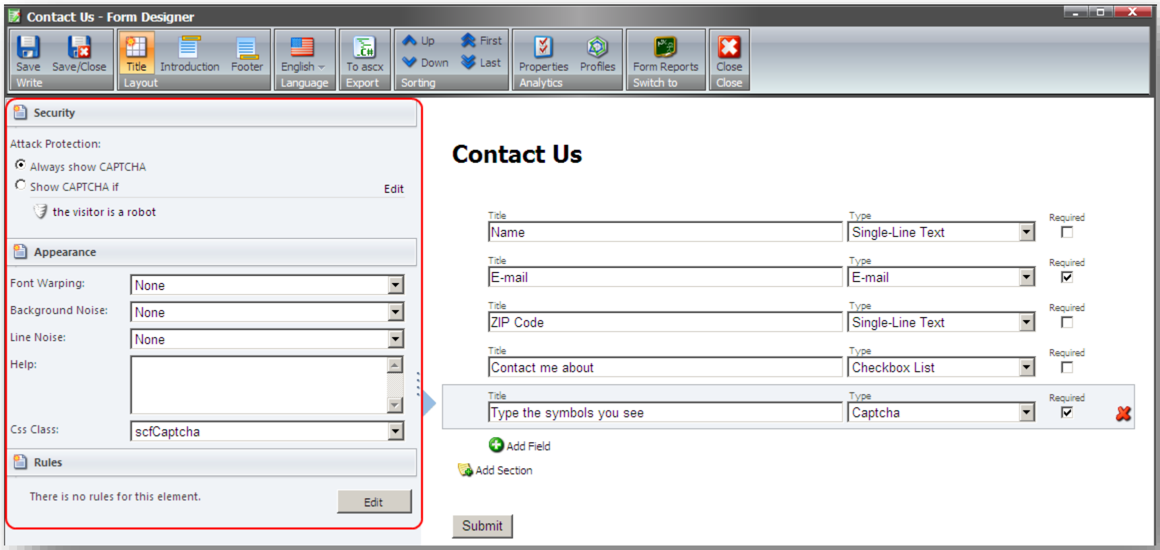
A **Section** is a special field type that you can use as a container for other fields.

Captcha

This field type contains two fields: an image field and a text confirmation field. The user should enter the text from the image field in the text field. This can be used to prevent robots registering on Web sites.



In the properties of the form field, you can set the amount of line noise, background noise, and font warp that should be used in the Captcha field:



Security

Attack Protection:

(●) Always show CAPTCHA
( ) Show CAPTCHA if — Edit
the visitor is a robot

Appearance

Font Warping:	None
Background Noise:	None
Line Noise:	None
Help:	
Css Class:	scfCaptcha

Rules

There is no rules for this element. — Edit

Contact Us

Title	Type	Required
Name	Single-Line Text	☐
E-mail	E-mail	☑
ZIP Code	Single-Line Text	☐
Contact me about	Checkbox List	☐
Type the symbols you see	Captcha	☑

Users configure the Captcha field using three options:

- The visitor is a robot.
- A suspicious visitor is detected.

  This option requires the users to enter the thresholds of times and minutes.
- Suspicious form activity detected.

  This option requires the users to enter the thresholds of times and minutes.

You can configure the range of the values that the users can enter in the last two options.

Password Confirmation

This field type contains two fields: a **Password** field and a **Confirm Password** field. These are used to create a password. Every character entered in these fields is masked.

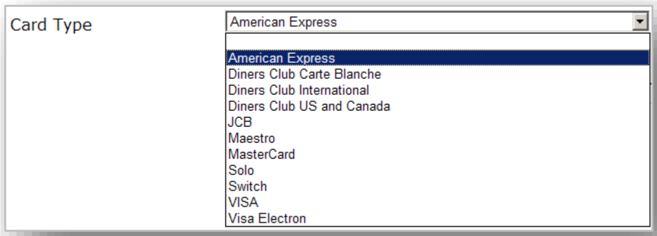
The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Complex/Password Confirmation` item contains the settings for this field type.

Credit Card

This field type contains two fields: one which allows the user to select a credit card type, and another which allows users to enter a credit card number.

This field type validates the credit card number with the possible number ranges and combinations allowed by the different types of credit card. You can specify that validation should be based on the Luhn formula or one of the following credit card types: American Express, Diners Club, Carte Blanche, Diners

Club International, Diners Club US and Canada, JCB, Maestro, MasterCard, Solo, Switch, Visa, Visa Electron.



The `/sitecore/System/Modules/Web Forms for Marketers/Settings/Field Types/Complex/Credit Card` item contains the settings for this field type.

A suspicious visitor is detected

In the `Sitecore.Forms.config` file, change the value of the *WFM.SessionThreshold* parameter. By default, the value is `2/1-100/60` — from 2 times in 1 minute to 100 times in 60 minutes.

Suspicious form activity detected

In the `Sitecore.Forms.config` file, change the value of the *WFM.ServerThreshold* parameter. By default, the value is `2/1-100/60` — from 2 times in 1 minute to 100 times in 60 minutes.

2.4.1 List Items

List field types include the following field types:

- Drop List
- List Box
- Radio List
- Checkbox List

You can specify the items displayed in a list by:

- Manually entering names
- Selecting Sitecore items
- Using XPath query
- Using Sitecore query
- Using fast query

The first two methods are used mainly by marketers. For more information about manually entering item names and selecting Sitecore items see the *Web Forms for Marketers User Guide*.

The last three methods can be used to find specific Sitecore items and use them as list field values.

List field types use the value-text concept. This means that list fields' values contain the text that is displayed to the user and the value that is stored in the database. The **Text** field of the list items is usually used to localize forms or to display user-friendly text.

Using XPath Query

This method is used to select Sitecore items with the help of XPath queries. Selected Sitecore items are used as list field values.

For more information about XPath Query, visit http://www.w3.org/TR/xpath/.

Using Sitecore Query

This method is used to select Sitecore items with the help of Sitecore queries. Selected Sitecore items are used as list field values.

For more information about Sitecore queries, visit the *Sitecore Developers Network*.

Using Fast Query

The **Using Fast query** method is used to select Sitecore items with the help of Fast queries. Selected Sitecore items are used as list field values.

For more information about Fast Query, visit the *Sitecore Developers Network*.

Localizing List Items

You can translate predefined values of the list items.

To localize list items that are specified using query methods:

1. Translate the same field (for example the **Display Name** field) for all Sitecore items that you will use as list items.
2. In the **Form Designer**, start editing the form and switch to the new language.
3. In the **List Items Wizard**, select the field that you translated as a **Text** value.

   The **Preview** is language specific, so you will see the translated items immediately.

2.5 Validations

The module contains predefined validations that allow users to add basic validations to fields, as well custom validations. These are explained in the *Web Forms for Marketers User Guide*.

The module also contains some built-in validations that are used for some of the form field types provided by default in the module.

The default validations are located under the `sitecore/System/modules/Web Forms for Marketers/settings/validation` item. The required field validator is located under the `/sitecore/system/Modules/Web Forms for Marketers/Settings/System/System Validation/NotEmpty` item.

The default validations are:

Validation	Description
Count chars	Checks the number of symbols in a string. You can set the minimum and maximum number of symbols.
Date	Checks whether or not the value entered is a date.
E-mail	Checks whether or not the value entered uses the format of an e-mail address.
Is Iso Date	Checks whether or not the value entered is an ISO date.
Number	Checks whether or not the values entered are numbers (negative numbers and integers are allowed).
Number range	Checks whether or not the values entered are within a specified range of numbers.
Regex pattern	Checks whether or not the values entered conform to a rule you specify.
Telephone	Checks whether or not the value entered is a valid telephone number.
SMS/MMS Telephone	Checks whether or not the value entered is a valid format for a text message or an MMS.
Credit card	Checks the validity of a credit card number based on the type of credit card.
Password-Confirmation	Compares the values entered in the **Password** and **Confirmation** fields
Captcha	Compares the text displayed on an image with the value entered by the user
Robot Protection	Checks if a current visitor is a robot (using the robot detection algorithm).
Suspicious Visitor	If a website visitor submits a web form several times in a short period of time, the visitor is considered suspicious
Suspicious Form Activity	Check whether a web form is submitted several times in short period of time by one or many users.

The module installs two validator templates:

- `/sitecore/Templates/Web Forms for Marketers/Validators/BaseValidator`.
- `/sitecore/Templates/Web Forms for Marketers/Validators/Regular Expression Validator`.

The *Regular Expression Validator* template inherits all the fields from the *BaseValidator* template and contains one more field: *Validation Expression*. Validations based on the *Regular Expression Validator* template use validation expressions. Validations based on the *BaseValidator* template use validations defined in classes.

A validator item contains the following fields:

Field	Description
Class	The full name of the class which handles the validation.
Assembly	The name of the assembly that contains the class.
Error Message	The text for the error message displayed in a *ValidationSummary* control when validation fails.
Text	The text displayed in the validation control when validation fails.
Static Display	The display behavior of the error message in a validation control: • *None* — the validation message is never displayed inline. • *Static* — space for the validation message is allocated in the page layout. • *Dynamic* — space for the validation message is dynamically added to the page if validation fails.
Enable Script Validation	Whether or not client-side validation is enabled.
Parameters	The additional parameters for the validation control.
Inner Control	Indicates the place where the validation control is added.
Validation Expression	Sets the regular expression assigned to be the validation criteria. (Only for validation items that use the *Regular Expression Validator*.)

2.5.1 How to Implement “Required” Checkbox Field Validator

In the Web Forms for Marketers module, the Checkbox field does not support the “required” validation rule. To make the Checkbox field “required”, follow one of the methods below.

Use the CheckboxList field

1. Add the Checkbox List field to a form.
2. Add only one item to the list.
3. Mark this Checkbox List as required.

Create a custom validator for the CheckboxList field

1. Create a class that is inherited from the `System.Web.UI.WebControls.BaseValidator` class. See a code sample:

```
class CheckboxValidation : BaseValidator
    {
```

```
                protected CheckBox ctrToValidate;
                protected CheckBox CheckBoxToValidate
                {
                    get
                    {
                        if (ctrToValidate == null)
                        {
                            ctrToValidate = base.FindControl(ControlToValidate) as CheckBox;
                        }
                        return ctrToValidate;
                    }
                }
                protected override bool ControlPropertiesValid()
                {
                    if (base.ControlToValidate == null || base.ControlToValidate.Length == 0)
                    {
                        throw new HttpException(string.Format("The ControlToValidate property of
'{0}' cannot be blank.", this.ID));
                    }
                    if (this.CheckBoxToValidate == null)
                    {
                        throw new HttpException(string.Format("The CheckBoxValidator can only
validate controls of type CheckBox."));
                    }
                    return true;
                }
                protected override bool EvaluateIsValid()
                {
        this.ErrorMessage = string.Format(this.ErrorMessage, "{0}", CheckBoxToValidate.Text);
                    //Validate whether checkbox is checked
                    return this.CheckBoxToValidate.Checked == true;
                }
            }
```

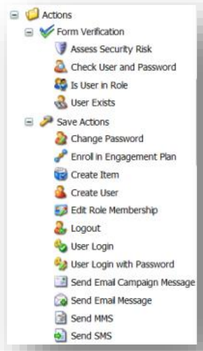
2. Create an item under the *sitecore/system/modules/web forms for marketers/settings/validation* folder. The item must be based on the *BaseValidator* template.
3. In the **Assembly** and **Class** fields, enter the appropriate values of the custom assembly.
4. In the **Error Message** field, enter this string: "*The {0} checkbox must be checked*"
5. In the **Text** field, enter appropriate message. If this field is blank its value is the same as the **Error Message** one.
6. Duplicate the */sitecore/system/modules/web forms for marketers/settings/field types/simple types/checkbox* item and rename it to **CheckboxRequired,** for instance.
   Note: In the **CheckboxRequired** item, do not select the "Required" checkbox.
7. In *the /sitecore/system/modules/web forms for marketers/settings/field types/simple types/CheckboxRequired* item, in the **Validation** field, select your custom validator.

2.6 Submit Actions

When a Web site visitor clicks Submit, three types of actions are performed:

- Form Verification
- Save Actions
- Success

All the actions are stored under the `/sitecore/System/Modules/Web Forms for Marketers/Settings/Actions` item.



Note:
If you upgrade from the previous version of the Web Forms for Marketers Module, your existing actions are moved to the **Save Actions** folder.

You can configure the following fields of an action:

- **Class** — the full name of the class that processes the form data on the server side.

- **Assembly** — the name of the assembly that contains the class referred to.

- **Parameters** — the parameters for the action irrespective of the parameters of the form. These are the global parameters, common to all the forms. For example, in the *Send Email Message* save action, this field is used to specify SMTP server to send e-mails.

- **Editor** — provides the application that you use to change the parameters of the action.

2.6.1 Form Verification

Form verification verifies the values that have been entered in one or more form fields. If form verification fails, the visitor is returned to the form and an error message is displayed. No other subsequent form verifications or save actions are performed.

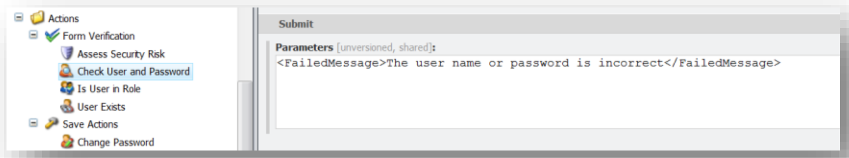
For more information about how to create a form verification action, see the section ***Error! Reference source not found.***.

Changing the Form Verification Error Message

You can set default form verification error message and localized one. For more information about how to set the localized error message, see the manual *Web Forms for Marketers User Guide*.

To set default error message:

1. In the **Content Editor**, select the form verification action that you want to edit.
2. In the **Parameters** field, enter an error message in the `<FailedMessage>` tag.



```
<FailedMessage>The user name or password is incorrect</FailedMessage>
```

2.6.2 Save Actions

Save actions can be assigned to a form and are executed when a visitor clicks **Submit** on a form. There are 17 save actions in the Web Forms for Marketers module, by default.

A save action is stored as a Sitecore item and executes an action implemented in a .net class. This class is specified in the *Class* and *Assembly* fields of the save action item. A save action item is based on the `/sitecore/templates/Web Forms for Marketers/Actions/Submit Action` template. Save actions items located under the `sitecore/ system/modules/web forms for marketers/settings/actions/save actions` item.

A save action item contains the following configuration fields:

Field	Description	Sample Field Value
Assembly	An assembly name that contains the associated class.	`Sitecore.Forms.Custom.dll`
Class	An associated class name including namespace.	`Sitecore.Form.Submit.SendMessage`

Parameters	Save action parameters. To use a parameter in the action, add a property to a custom action class, and name this property as parameter named.	`<DefaultDomain>extranet</DefaultDomain>`
		`If the action class contains the` *`DefaultDomain`* `property, this property is initialized with the extranet value:`
		`/// <summary>`
		`/// Gets or sets the default domain.`
		`/// </summary>`
		`/// <value>The default domain.</value> public string DefaultDomain { get; set; }`

Localized Parameters	Similar to **Parameters**. The only difference is that this field is not shared so a parameter can be localized.	`<DefaultDomain>extranet</DefaultDomain>`
		`If the action class contains the` *`DefaultDomain`* `property, this property is initialized with the extranet value:`
		`/// <summary>`
		`/// Gets or sets the default domain.`
		`/// </summary>`
		`/// <value>The default domain.</value> public string DefaultDomain { get; set; }`

Field	Description	Sample Field Value
Client Action	This field is only used in the staging environment. If this check box is cleared, this save action is transferred from the Slave to the Master server and is performed there. If this check box is selected, a save action is performed on the Slave server.	Selected.
Editor	A control that Sitecore user uses when they edit the parameters for the save action in the Form Designer.	`control:Forms.MappingFields`
QueryString	Additional settings for the editor.	`Fields=Login|Login,Password|Password`

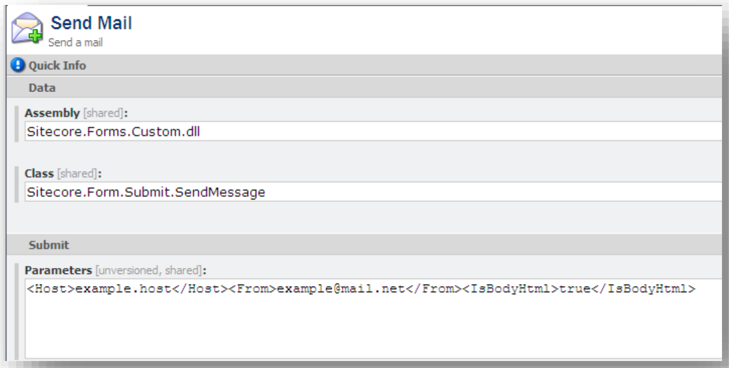
You can create a custom save action. For more information about creating a save action, see the Creating a Save Action section.

There are three save actions that send notifications to a user: *Send Email Message*, *Send SMS* and *Send MMS*. All these save actions use default SMTP server settings specified in the `web.config` file. You can specify different SMTP server settings for each save action.

Send Email Message

To use different SMTP settings for this action, in the **Content Editor**, navigate to *Sitecore/System/Modules/Web Forms for Marketers/Settings/Actions/Save Actions/Send Email Message*, in the **Parameters** field, set the appropriate values.

For example, to use *Host* and *From* parameters that are different from ones defined in the `web.config` file, enter the following code into the **Parameters** field:



Assembly [shared]:
Sitecore.Forms.Custom.dll

Class [shared]:
Sitecore.Form.Submit.SendMessage

Parameters [unversioned, shared]:
```
<Host>example.host</Host><From>example@mail.net</From><IsBodyHtml>true</IsBodyHtml>
```

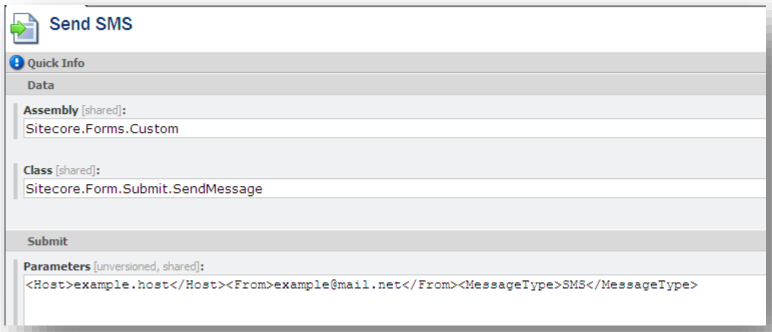
You can also specify credentials for the SMTP server in the **Parameters** field:

```
        <Login>login</Login><Password>pass</Password>
```

Send SMS

To use different SMTP settings for this action, in the **Content Editor**, navigate to *Sitecore/System/Modules/Web Forms for Marketers/Settings/Actions/Save Actions/Send SMS*, in the **Parameters** field, set the appropriate values.

For example, to use `Host` and `From` parameters that are different from one defined in the `web.config` file, enter the following code into the **Parameters** field:



Assembly [shared]:
Sitecore.Forms.Custom

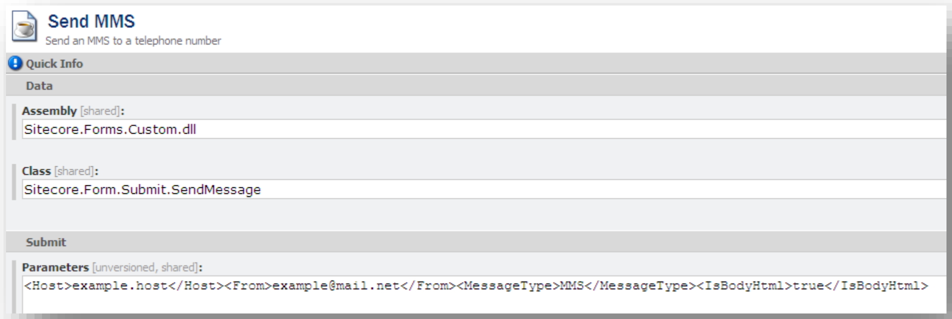
Class [shared]:
Sitecore.Form.Submit.SendMessage

Parameters [unversioned, shared]:
```
<Host>example.host</Host><From>example@mail.net</From><MessageType>SMS</MessageType>
```

Send MMS

To use different SMTP settings for this action, in the **Content Editor**, navigate to *Sitecore/System/Modules/Web Forms for Marketers/Settings/Actions/Save Actions/Send MMS*, in the **Parameters** field, set the appropriate values.

For example, to use `Host` and `From` parameters that are different from one defined in the `web.config` file, enter the following code into the **Parameters** field:



Assembly [shared]:
Sitecore.Forms.Custom.dll

Class [shared]:
Sitecore.Form.Submit.SendMessage

Parameters [unversioned, shared]:
```
<Host>example.host</Host><From>example@mail.net</From><MessageType>MMS</MessageType><IsBodyHtml>true</IsBodyHtml>
```

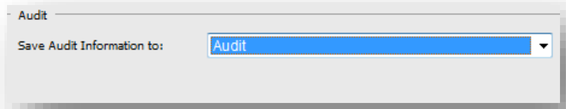
2.6.3 Success

This action allows you to select either a website page or a message which is presented to a Web site visitor after they successfully submit a form. This is the final action performed in a form submission. The pipeline attached to this is called `successAction` pipeline.

2.6.4 Auditing information

Some default save actions create or edit users or roles in Sitecore's security model. These are referred to as *security actions*. These include the *Create User, Edit Role Membership, User Login*, and *User Login* *with Password* save actions. As all of these can affect user information, the ability to register audit information in the user profile to record what actions have been performed is useful.

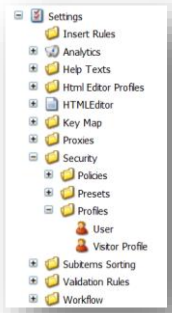
The security actions all have a **Save Audit Information to:** dropdown list which list the field in the user profile to which audit information can be written.



By default all rich text, html, text, memo, multi-line text, and single-line text fields can be used to register audit information. The field types in which audit information can be registered can be configured using the `WFM.AuditAllowedTypes` setting in the `Sitecore.Forms.config` file:

```
<setting name="WFM.AuditAllowedTypes" value="|Rich Text|html|text|Multi-Line
Text|Single-Line Text|memo|" />
```

All user profiles are items in the Core database in the */sitecore/system/Settings/Security/Profiles* folder:



The *Visitor profile* is used by default. Each form item has a reference to this user profile in the *User Profile* field.

2.7 Reports

You can see information how Web site visitors interact with forms in the various reports that come with the Web Form module.

The Web Forms for Marketers module supports the creation of the form dropout reports which contain information about users who did not successfully submit forms. The functionality is achieved using AJAX features by tracking each individual field and recording user input in the *Analytics* database.

System events are recorded in the session trail. For more information about system events, see the *Events and the (Visit* Details) Session Trail section.

Web reports use aggregation and reporting API as part of DMS functionality.

To open the form reports:

1. Log in to the Sitecore desktop
2. Click *Sitecore*, *All Applications*, *Web Forms for Marketers*, *Form Reports*
3. Select the form for which you want to view reports.

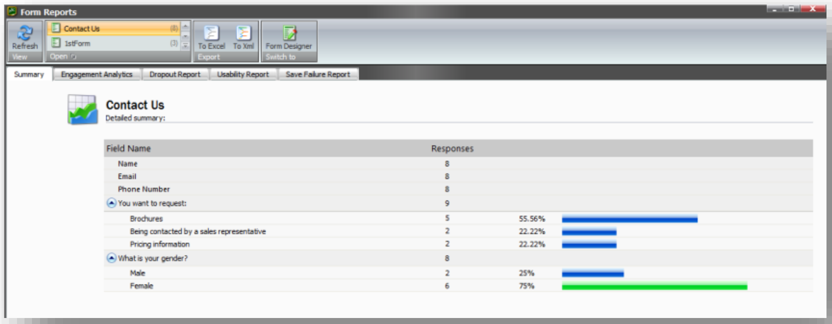
Alternatively, in the **Content Editor**, select the form under the `/sitecore/System/Modules/Web Forms for Marketers/Website` item and click **Form Reports**.

Note
You will only be able to open the **Form Reports** if the form contains marketing analytics statistics. If a form is created with the **Enable Marketing Analytics** option disabled, the **Form Reports** button in the Form Designer or Content Editor is disabled (grayed out).

The following reports are available:

- Summary
- Engagement Analytics
- Dropout Report
- Usability Report
- Save Failure Report

The form reports are displayed on the different tabs in the **Form Reports** window:



Contact Us
Detailed summary:

Field Name	Responses		
Name	8		
Email	8		
Phone Number	8		
You want to request:	9		
Brochures	5	55.56%	▬
Being contacted by a sales representative	2	22.22%	▬
Pricing information	2	22.22%	▬
What is your gender?	8		
Male	2	25%	▬
Female	6	75%	▬

All the reports display the last 200 records by default. To change this number, open the report in the Web Reports Designer and change the value in the appropriate SQL query.

2.7.1 Supported Databases

The Web Forms for Marketers module reports support the following database servers:

Report	Supported Database
Summary	Reporting API
Engagement Analytics	Reporting API
Dropout Report and sub-reports	Reporting API
Usability Report and sub-reports	Reporting API
Save Failures Report	Reporting API

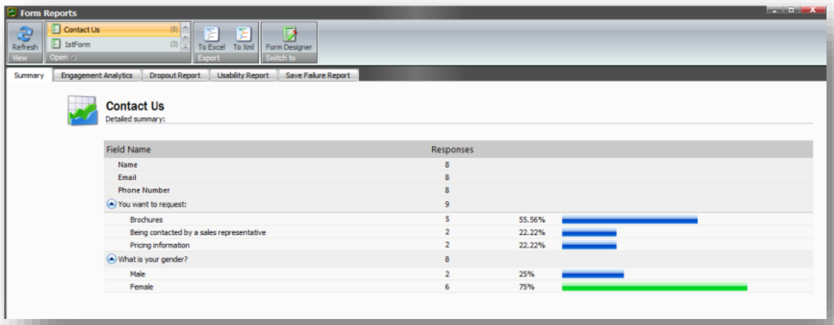
For more information about the Reporting API, see the documentation on the DMS API.

2.7.2 Data Request Timeouts

By default the Data report request timeout is 180 seconds. You can modify this value by changing the `WFM.CommandTimeout` parameter in the `\WebSite\App_Config\Include\Sitecore.Forms.config` file.

2.7.3 Summary

All records are displayed in this report by default.



Contact Us
Detailed summary:

Field Name	Responses		
Name	8		
Email	8		
Phone Number	8		
You want to request:	9		
Brochures	5	55.56%	
Being contacted by a sales representative	2	22.22%	
Pricing information	2	22.22%	
What is your gender?	8		
Male	2	25%	
Female	6	75%	

If the value of any field exceeds 80% then the blue stripe turns green by default. You can modify this value by changing the `WFM.RelevantScale` parameter in the `\WebSite\App_Config\Include\Sitecore.Forms.config` file.

2.8 Configuring a User's Access to the Module

To configure the access that a user has to the features in the module, an administrator can assign the following roles to the user:

- Sitecore Client Form Author
- Sitecore Client Developing
- Analytics Maintaining
- Analytics Reporting
- Sitecore Marketer Form Author
- Sitecore Client Securing

To give the user minimum access rights to the module, assign the *Sitecore Client Form Author* role to the user.

To give the user access to all the features in the module, assign the following roles to the user:

- Sitecore Marketer Form Author
- Sitecore Client Developing
- Sitecore Client Securing

2.8.1 The Security Roles for Web Forms

Here is a short description of the security roles that have been created for the Web Forms for Marketers module:

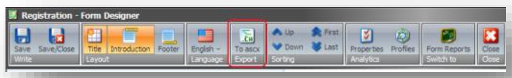
Sitecore Client Form Author

The *Sitecore Client Form Author* role gives the user access to the minimum features of the Web Forms for Marketers module. All the other roles expand the user's access rights. This role allows the user to:

- Insert a new form.
- Edit an existing form.
- View the *Summary* report.

Sitecore Client Developing

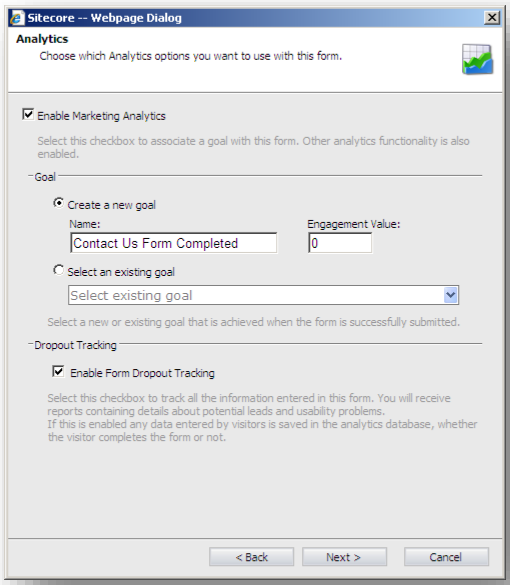
The *Sitecore Client Developing* role allows the user to use the **Export to ASCX** button in the **Form Designer**:



Analytics Maintaining

The **Analytics Maintaining** role allows the user to:

- Use the **Analytics** page in the **Create a New Form** wizard:



Sitecore -- Webpage Dialog

Analytics

Choose which Analytics options you want to use with this form.

[✓] Enable Marketing Analytics

Select this checkbox to associate a goal with this form. Other analytics functionality is also enabled.

Goal

(●) Create a new goal

Name: Contact Us Form Completed

Engagement Value: 0

( ) Select an existing goal

Select existing goal

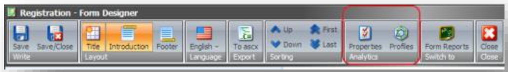
Select a new or existing goal that is achieved when the form is successfully submitted.

Dropout Tracking

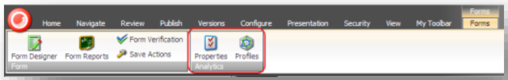
[✓] Enable Form Dropout Tracking

Select this checkbox to track all the information entered in this form. You will receive reports containing details about potential leads and usability problems.
If this is enabled any data entered by visitors is saved in the analytics database, whether the visitor completes the form or not.

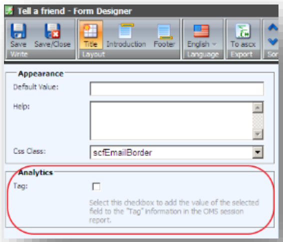
- Use the **Analytics** section on the **Form Designer** ribbon:



- Use the **Analytics** section on the **Content Editor** ribbon:



- Use tags:



Appearance

Default Value:

Help:

Css Class: scfEmailBorder

Analytics

Tag: [ ]

Select this checkbox to add the value of the selected field to the "Tag" information in the OMS session report.

Analytics Reporting

The *Analytics Reporting* role allows the user to:

- View the Dropout report.
- View the Usability report.
- View the Save Failures report.

Sitecore Marketer Form Author

The *Sitecore Marketer Form Author* role inherits access rights from the following roles:

- Sitecore Client Form Author
- Analytics Maintaining
- Analytics Reporting

Members of the *Sitecore Marketer Form Author* role have all the rights assigned to these roles.

Sitecore Client Securing

The *Sitecore Client Securing* role allows the user to:

- Edit the *Create User* save action.
- Edit the *Edit Role Membership* save action.
- Edit the *Change Password* save action.

2.9 Events and the (Visit Details) Session Trail

The (visit details) session trail is an Sitecore Engagement Analytics feature which records all the activities a user has performed on a Web site, including what pages they have visited and when.

The session trail reports also list the events that are triggered when a visitor fills in a form. Those events are standard Sitecore Engagement Analytics page events that can be used for development purposes. The reports list both field events and form events.

For all the events, the **Datakey** field in the Analytics database equals the Form GUID, the exceptions are: Form check action error, Form submit, and Form conversion events.

Several field events can occur in the same field during the same attempt to submit the form, as users change the information in the field.

The events that are listed in the form reports are:

Field Completed

This event is triggered when a field on a form is completed and tabbed or clicked out of. This is done with AJAX.

Field Not Completed

This event is triggered when validation of a required field fails because the field has not been filled in by the visitor.

Field Out of Boundary

This event is triggered when validation of a field fails because the value entered falls outside the boundaries defined for the field. This event is used for the Min and Max Length of Text and Password fields, as well as for Date and Number fields.

Form Check Action Error

This event is triggered when a Check Action fails. This is not a failure, but an event. If a Check Action fails, the visitor is returned to the form. No other Check Actions or save actions are initiated.

Form Save Action Failure

This event is triggered when a save action fails. This is a failure and not an event — the *IsFailure* property is enabled).

Form Submit

This event is triggered when a visitor clicks the **Submit** button or presses ENTER. This indicates an attempt to submit the form.

Invalid Field Syntax

This event is triggered when validation of a field fails because the data entered in the field fails a particular syntax check. The event is used for the following field validations: Regular Expressions in Text and Password types, and Email fields.

Submit Success

This event is triggered when a Submit action does not return an error. The event is written in along with the Form Conversion event and is primarily used to facilitate the SQL statements required for the *Form Dropout*, *Form Usability*, and *Form Failures* reports.

Form Conversion

The event is triggered when a form is successfully submitted and is triggered after the Form Submit event. It also indicates that the goal associated with the form has been successfully completed.

Form Begin (system event)

This is a system event that is not displayed in the reports. Form Begin is triggered when the user starts working with a web form (fill in a field or click the Submit button).

Important
The Form Begin event is critical to the reporting. We do not recommend you trigger it because this can affect form dropout reporting.

Form Dropout (system event)

This is a system event that is not displayed in the reports. Form Dropout is triggered when the user session is expired and the user did not successfully submitted the web form.

Important
The Form Dropout event is critical to the reporting. We do not recommend you trigger it because this can affect form dropout reporting.

2.10 Multi-site Implementation

The Web Forms for Marketers module supports multi-site environments. This means that administrators can define different form locations and settings for different Web sites. This is done in the `formsRoot` attribute in the definition of the site in the `.config` files.

The value of this attribute is a Sitecore path which defines:

- The folder that stores the forms on the current site.
- The appearance and color settings for forms on the current site.
- The access rights.

The `formsRoot` parameter must contain either an item path or the ID of the target item. The target item must be based on the `/sitecore/Templates/Web Forms for Marketers/Forms Folder` template.

For example, this is how the `formsRoot` parameter is defined in `web.config` file:

```
<sites>
 <site
     name="samplesite"
     virtualFolder="/"
     physicalFolder="/"
     rootPath="/sitecore/content"
     startItem="/forms" database="web" domain="extranet"
     formsRoot="/sitecore/System/modules/Web Forms for Marketers/local forms"
     ...
```

This can be defined in the `Sitecore.Forms.config` file using the ID:

```
 <site name="website">
   <patch:attribute name="formsRoot">
     {F1F7AAB6-C8CE-422F-A214-F610C109FA63}
   </patch:attribute>
 </site>
</sites>
```

We recommend that you define the `formsRoot` parameter in the `/App_Config/Include/Sitecore.Forms.config` file. This approach ensures that there are no duplicated values.

When the `formsRoot` attribute is not defined for a Web site, new forms are stored in the `/sitecore/System/Modules/Web Forms for Marketers/Local Forms` folder.

2.11 Multi-server Environment

You can scale the Web Forms for Marketers module by configuring a multi-server environment. In such a configuration the module partially delegates executing of save actions from the content delivery server to the master one.

To configure a multi-server environment:

1. Install the module on the master server. For more information about installing the module, go to the Web Forms for Marketers section on the Sitecore Developer Network.
2. Deploy the module on all content delivery servers.

2.11.1 Deploying the Module on the Content Delivery Server

When the module is installed on the master server, deploy it on all content delivery servers.

To deploy the module on the content delivery server:

1. Publish the Master database to all the content delivery servers. For more information about publishing the Master database, see the CMS Scaling Guide.
2. Unzip the Web Forms for Marketers CD 2.5 rev. XXXXXX.zip package to the \Website folder. This archive contains the following files:
   - Bin\sitecore.forms.core.dll
   - Bin\Sitecore.Forms.Custom.dll
   - Bin\MSCaptcha.dll
   - App_Config\Include\Sitecore.Forms.config
   - App_Config\Include\Sitecore.Forms.Xtune.CD.config
   - App_Config\Include\Captcha.config
   - Sitecore modules\shell\Web Forms for Marketers\Themes\*.*
   - Sitecore modules\web\Web Forms for Marketers\scripts\*.*
   - Sitecore modules\web\Web Forms for Marketers\Tracking.aspx
   - Sitecore modules\web\Web Forms for Marketers\control\*.ascx
   - Sitecore modules\web\Web Forms for Marketers\UI\UserControl\*.ascx
3. Add the following connection string to the `connectionstrings.config` file on all content delivery servers:

```
        <add name="remoteWfmService"
connectionString="url=http://[masterserver]/sitecore%20modules/shell/Web%20Forms%20for%20Marketer
s/Staging/WfmService.asmx;user=[domain\username];password=[password];timeout=60000" />
```

   Where

   `[masterserver]` – IP or the host name of the master server.

   `[domain\username]` – Sitecore user (full name). The required access rights depend on the items that the form is using.

   `[password]` – a user's password.

Note
After you added the *remoteWfmService* connection string to the `connectionstrings.config` file on all content delivery servers, all save actions with the cleared **Client Action** check box are executed on the content management server instead of the current server. All uploaded files are uploaded to the content management server too. For more information about the **Client Action** check box, see the section *Save Actions.*

2.11.2 Deploying the Module on the Processing Server

To deploy the module on the processing server:

1. Configure the processing server according to the CMS Scaling Guide.
2. Unzip the Web Forms for Marketers Processing 2.5 rev. XXXXXX.zip package to the \Website folder. This archive contains the following files:
    - Bin\sitecore.forms.core.dll
    - App_Config\Include\Sitecore.Forms.config
    - App_Config\Include\Sitecore.Forms.Xtune.Processing.config

2.12 Running Web Forms in Live Mode

Sitecore supports running a Web site directly from the master database, referred to as — "running in live mode". Running in live mode eliminates the need to publish content and is similar to viewing a site in the Preview client. A Web site that is configured to run in live mode acts exactly like a normal Web site. Live mode respects all publishing restrictions and workflows in the same way that a default Web site supports these features.

To run the Web Forms for Marketers module in live mode, you must edit the `web.config` file:

1. In the `web.config` file, find the relevant `<site>` section and change `database="web"` to `database="master"`.
2. Find the `<modules_website>` section and change `database="web"` to `database="master"`.

   The `<sites>` section of your `web.config` file should look like this:

   ```
   <sites>
   ...
         <site name="modules_website"...database="master".../>

         <site name="website"...database="master".../>
   ...
   </sites>
   ```

Chapter 3

Web Forms Developer’s Notes

This chapter contains information for developers on how to customize the Web Forms for Marketers module.

This chapter contains the following sections:

- How to Add an ASCX Control to the Page
- How to Re-Install the Module
- How to Uninstall the Module

3.1 Initial Settings

3.1.1 CAPTCHA

The Web Forms for Marketers module lets you implement CAPTCHA so that it works in a multi-instance environment.

If you want to implement this, you must manually register CAPTCHA image and audio handlers in the `web.config` file.

Note
You must register CAPTCHA on each content delivery application instance.

In the `web.config` file, in the `handlers` and `httpHandlers` sections, in the `add` nodes, add the corresponding handler references:

```
        <configuration>
          <system.webServer>
            <handlers>
        <add name="CaptchaImage" verb="*" path="CaptchaImage.axd"
type="Sitecore.Form.Core.Pipeline.RequestProcessor.CaptchaResolver, Sitecore.Forms.Core" />
        <add name="CaptchaAudio" verb="*" path="CaptchaAudio.axd"
type="Sitecore.Form.Core.Pipeline.RequestProcessor.CaptchaResolver, Sitecore.Forms.Core" />
            </handlers>
          </system.webServer>

          <system.web>
            <httpHandlers>
        <add name="CaptchaImage" verb="*" path="CaptchaImage.axd"
type="Sitecore.Form.Core.Pipeline.RequestProcessor.CaptchaResolver, Sitecore.Forms.Core" />
        <add name="CaptchaAudio" verb="*" path="CaptchaAudio.axd"
type="Sitecore.Form.Core.Pipeline.RequestProcessor.CaptchaResolver, Sitecore.Forms.Core" />
            </httpHandlers>
          </system.web>
       </configuration>
```

3.2 Sitecore.Forms.config

This section describes the main configuration parameters of the `Sitecore.Forms.config` file, located in the `\Website\App_Config\Include` folder.

3.2.1 Events

There is only one web forms specific event in the `Sitecore.Forms.config` file – `forms:save`. This event is raised when a web form is submitted before save actions are performed.

3.2.2 Pipelines

The module uses the following pipelines:

Pipeline	Description
formUploadFile	Runs when a file is uploaded using the FileUpload field of a web form.
successAction	Runs when a web form is submitted successfully. Default processors redirect the user to the success page or generate the success message.
errorSubmit	Runs when an exception occurs when the user submits a web form.
errorSave	Runs when an exception occurs when the module processes a save action.
errorCheck	Runs when an exception occurs when the module processes a verification action.
exportToXml	Runs when the module exports web forms data to an XML document from the Data Viewer.
exportToExcel	Runs when the module exports web forms data to a Microsoft Excel document from the Data Viewer.
parseAscx	Runs when the module launches the **Convert to ascx** dialog box. This pipeline parses a web form and generates the output .ascx control.
exportToAscx	Runs when the user clicks **Download** in the **Convert to ascx** dialog box. This pipeline generates an .ascx control for downloading.
auditRender	Writes audit information to the user profile when the module performs a save action related to security, for example the Create User save action.
processMessage	Runs when the module generates an email message for the Send Email Message save action.

3.2.3 Commands

This section contains UI commands specific to the Web Forms for Marketers module.

For example, when the user in the Content Editor, on the Presentation tab, clicks the **Insert** button the `forms:insert` command is performed.

3.2.4 Settings

This section contains the settings that you can use to configure the Web Forms for Marketers module. Each setting contains a short description that will help to understand how this setting works.

3.3 Modifying Module Behavior Using Custom Processors

This section contains sample actions that will show you how to modify module behavior using custom processors.

3.3.1 How to Configure an Email Message Using the ProcessMessage Pipeline

In this example, we describe how to configure an email message of the Send Email Message save action using the `ProcessMessage` pipeline.

To configure an email message, perform the following actions:

1. Create a processor class using the following sample code:

```
        using Sitecore.Form.Core.Pipelines.ProcessMessage;
        // This processor adds a note to the end of the email body
          public class AddTextToBody
          {

            public void Process(ProcessMessageArgs args)
            {
              string additionalText = "<p>This message was sent using the Sitecore Web Forms for
Marketers module.</p>";
              args.Mail.Append(additionalText);
              /*
               * it's also possible to modify SUBJECT, TO, CC and BCC message fields
               * args.Subject.Append(" subject text");
               * args.To.Append("; secondrecipient@mail.net");
               * args.CC.Append("; secondrecipient@mail.net");
               * args.BCC.Append("; secondrecipient@mail.net");
               * args.From = "sender@mail.net";
               */
            }
          }
```

2. Register the new processor in the `Sitecore.Forms.config` file, before the `Sitecore.Form.Core.Pipelines.ProcessMessage.ProcessMessage` processor:

```
              <processMessage>
                …
                <processor type="YourNamespace.AddTextToBody,YourAssemblyName" />
                <processor type="Sitecore.Form.Core.Pipelines.ProcessMessage.ProcessMessage,
Sitecore.Forms.Core" method="SendEmail"/>
              </processMessage>
```

3.3.2 How to Send SMS/MMS Using a Custom Processor

The *Send MMS* and the *Send SMS* save actions delegate sending messages to MMS/SMS gateways through a SMTP server.

You can use the `processMessage` pipeline to change this behavior and send messages, for example, through a third party paid Web service.

1. Create a new processor using the following sample code:

```
        namespace Sitecore.Form.Core.Pipelines.ProcessMessage
        {
          using System.IO;
          using System.Net;

          public class SendSMSorMMS
          {
            public void Process(ProcessMessageArgs args)
            {
              if (args.MessageType == MessageType.MMS || args.MessageType == MessageType.SMS)
              {
                WebClient wc = new WebClient();
                wc.Credentials = (NetworkCredential)args.Credentials;

                wc.QueryString.Add("sendto", args.Recipient);
                wc.QueryString.Add("message", args.Mail.ToString());
                if (!string.IsNullOrEmpty(args.From))
                {
                  wc.QueryString.Add("from", args.From);
                }
                using (Stream responseStream = wc.OpenRead("https://3rdparty.smsormms.com/"))
                {
                  using (StreamReader responseReader = new StreamReader(responseStream))
                  {
                    responseReader.ReadToEnd();
                    responseReader.Close();
                    responseStream.Close();
                  }
                }
              }
            }
          }
        }
```

2. Register the new processor in the `Sitecore.Forms.config` file:

```
              <processMessage>
                <processor type="Sitecore.Form.Core.Pipelines.ProcessMessage.ProcessMessage,
Sitecore.Forms.Core" method="ExpandLinks"/>
                <processor type="Sitecore.Form.Core.Pipelines.ProcessMessage.ProcessMessage,
Sitecore.Forms.Core" method="ExpandTokens"/>
                <processor type="Sitecore.Form.Core.Pipelines.ProcessMessage.ProcessMessage,
Sitecore.Forms.Core" method="AddHostToItemLink"/>
                <processor type="Sitecore.Form.Core.Pipelines.ProcessMessage.ProcessMessage,
Sitecore.Forms.Core" method="AddHostToMediaItem"/>
                <processor type="Sitecore.Form.Core.Pipelines.ProcessMessage.ProcessMessage,
Sitecore.Forms.Core" method="AddAttachments"/>
                <processor type="Sitecore.Form.Core.Pipelines.ProcessMessage.ProcessMessage,
Sitecore.Forms.Core" method="BuildToFromRecipient"/>
                <processor type="Sitecore.Form.Core.Pipelines.ProcessMessage.SendSMSorMMS,
MyAssembly"/>
                <processor type="Sitecore.Form.Core.Pipelines.ProcessMessage.ProcessMessage,
Sitecore.Forms.Core" method="SendEmail"/>
              </processMessage>
```

3.3.3 How to Forbid Users to Upload Large Files via the File Upload field

In this example, we describe how to forbid users to upload files larger than 10 MB via the File Upload form field.

To forbid uploading large file via the File Upload field, perform the following actions:

1. Create a new processor class using the following sample code:

```
        using Sitecore.Form.Core.Pipelines.FormUploadFile;

          public class UploadingLimitation
          {
            public void Process(FormUploadFileArgs args)
            {
              int size = 10485760; // == 10 Mb
              if (args.File.Data.Length > size)
              {
                Sitecore.Diagnostics.Log.Info(string.Format("User {0} tried to upload a file
 larger than 10 Mb. The file name is {1}",
                  Sitecore.Context.User.Name,
                  args.File.FileName), this);
                args.AbortPipeline();
              }
            }
          }
```

2. Register the new processor in the `Sitecore.Forms.config` file:

```
       <formUploadFile>
               <processor type="YourNamespace. UploadingLimitation, YourAssemblyName"/>
               …
       </formUploadFile>
```

After this solution is implemented, when the user tries to upload a file larger than 10 MB via the File Upload form field, the module does not let the user do that and corresponding message is saved to log files.

3.4 How to Insert a Web Form on a Web Page

Users can add a web form to a web page using the Page Editor. Developers and administrators can add a web form to a web page in the following ways:

- Insert a web form as a Sitecore standard rendering.
- Insert a web form as a web control.
- Insert a web form using the code-behind class.

Inserting a web form as a Sitecore rendering

A web form is a Sitecore rendering so you can insert it in the Content Editor:

1. In the Content Editor, select the item, where you want to add a web form.
2. On the ribbon, in the **Presentation** tab, click **Details**.
3. In the **Layout Details** dialog box, click **Edit** for the appropriate device.
4. In the **Device Editor** dialog box, add a new control: *Renderings/Modules/Web Forms for Marketers/Form*.
5. Edit the added control. In the **FormID** field, select the appropriate web form.

Inserting a web form as a web control

You can add a web form to a layout statically:

1. Open an `*.aspx` or `*.ascx` file.
2. Register a tag prefix for the web forms namespace:

```
        <%@ Register TagPrefix="wffm" Namespace="Sitecore.Form.Core.Renderings"
Assembly="Sitecore.Forms.Core" %>
```

3. Add the `FormRenderer` tag:

```
        <wffm:FormRender  FormID="<id of the form item>" runat="server"/>
```

Inserting a web form using the code-behind class

You can add different web forms depending on different conditions using the code-behind class. Use the following sample code in the code-behind class:

```
        FormRender fr = new FormRender();
        fr.FormID = "5D9E85F3-5E03-49A7-A136-93269DEA22A7";//form item id
        fr.FormTemplate = "/sitecore modules/web/Web Forms for
Marketers/Control/SitecoreSimpleFormAscx.ascx";
        Sitecore.Context.Page.GetPlaceholder("main").Controls.Add(fr);
```

This sample code inserts a web form with the specified `ID` to the `Main` placeholder.

3.5 How to Configure a Data Provider

By default, the module uses the Mongo Analytics data provider.

The data provider is configured in the `formsDataProviders` section in the `Sitecore.Forms.config` file.

```
        <formsDataProviders>
          <main type="Sitecore.Forms.Core.Analytics.Data.AnalyticsDataProvider"></main>
              <!-- Custom data storage used to store forms data to a database other than XDb -->
              <!--<custom type="Sitecore.Forms.Sample.CustomDataProvider,CustomFormsProvider">
                 <param desc="connection string">user id=sa;password=12345;Data
Source=localhost;AttachDBFilename=F:\kontiki\Data\Sitecore_WebForms.mdf</param>
              </custom>-->
        </formsDataProviders>
```

You can add additional data providers for other databases, too. See the example above for more information about how to add an additional data provider for a database. When you submit a form, the `InsertForm` method is executed for all available data providers. Set the `FormDataManager.ProviderContext` property to specify what provider must be used for retrieving the forms data. By default, the `ProviderContext` is always assigned the `main` value. For example, if you want to have a form retrieved from custom database, set `ProviderContext` to `custom`.

3.6 How to Extend/Override Standard Functionality

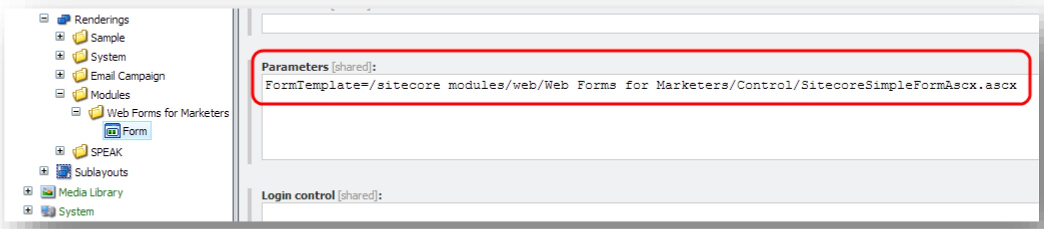
The Web Forms for Marketers module allows you to extend the existing functionality.

3.6.1 Form Rendering

The module displays web forms on the website using the `Form` rendering. You can find it in the Content Editor, in the */sitecore/layout/Renderings/Modules/Web Forms for Marketers/* item. When the user clicks **Insert** to add a web form, the module automatically adds the `Form` rendering to the presentation of the current item.

The `Form` item contains some configuration fields, for instance:

- Tag – rendering class that defines its logic.
- Namespace – class namespace.
- Assembly – class assembly.
- Parameters – by default, this field specifies the path to the ASP.NET user control file (`*.ascx`) that represents a web form: `FormTemplate=/sitecore modules/web/Web Forms for Marketers/Control/SitecoreSimpleFormAscx.ascx`.



Parameters [shared]:

```
FormTemplate=/sitecore modules/web/Web Forms for Marketers/Control/SitecoreSimpleFormAscx.ascx
```

Login control [shared]:

You can change a layout of the web forms, add client scripts, and affect the web form rendering and life cycle. To perform such a customization that will affect all the web forms on the website, edit the `SitecoreSimpleFormAscx.ascx` file.

To change the global web forms logic:

- Create a new class that is inherited from the `Sitecore.Form.Web.UI.Controls.SitecoreSimpleFormAscx` class.
- Override appropriate methods.
- In the `SitecoreSimpleFormAscx.ascx` control, in the `Inherits` attribute, replace the standard class with the created one.

Important
We recommend that you only edit `the SitecoreSimpleFormAscx.ascx` file if there is no other way to override the standard functionality. This method might cause some troubles when you will update the module.

Example. How to Add a Logo to all the Web Forms on the Website

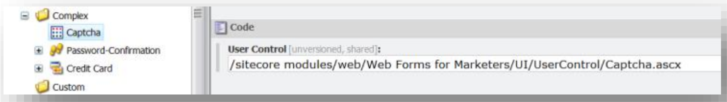
In the `/sitecore modules/web/Web Forms for Marketers/Control/SitecoreSimpleFormAscx.ascx` file, before the `<wfm:FormTitle>` tag, add the `<img>` tag pointing to the logo image, for example:

```
        ….
        <img src="/images/logo.png" alt="Logo" style=" float:right;" />
        <wfm:FormTitle ID="title" runat="server"/>
        ….
```

Now all the web forms on the website will contain a logo.

3.6.2 Field Controls

The Web Forms for Marketers module supports `ascx` field controls. You should type the path to an `.ascx` control to use it in the form.



User Control [unversioned, shared]:
/sitecore modules/web/Web Forms for Marketers/UI/UserControl/Captcha.ascx

You can change the appearance of the following `.ascx` field controls:

- Password-Confirmation
- Credit-Card
- Captcha

These are located in the `sitecore\modules\Web\Web Forms for Marketers\UI` folder.

3.6.3 Field Actions

Web Forms for Marketers allows you to personalize web forms by configuring actions that will be executed when the specified conditions are met. A set of conditions and corresponding actions is called rule. The module contains the predefined set of actions. You can implement a customs field action.

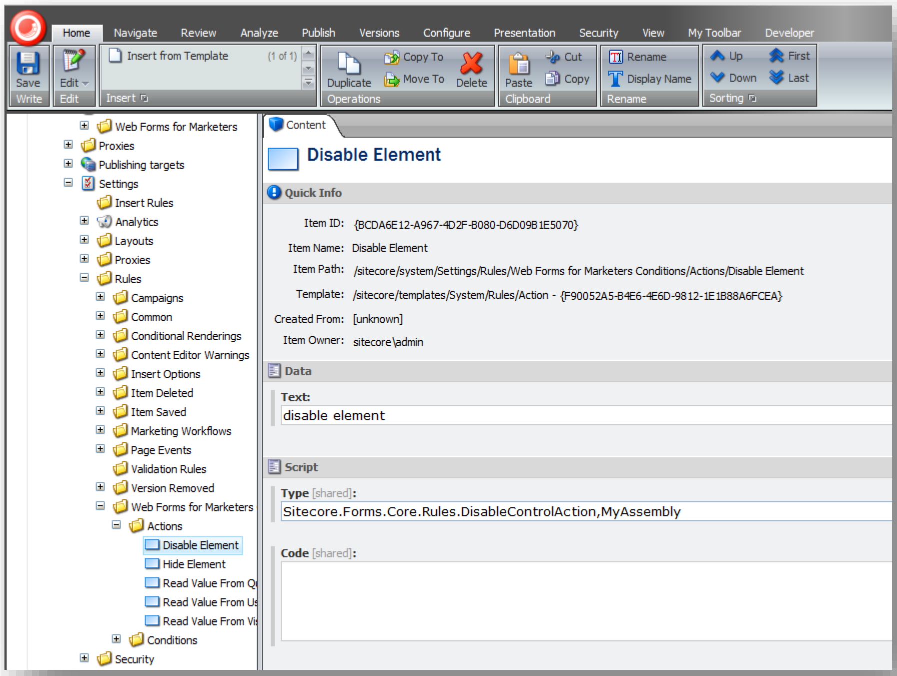
For example, you want to implement an action that will disable input in the field. This might be useful if your web form is designed to update user profiles, shop orders and so on. The web form displays a field containing its value but does not allow editing it because, for example, *user ID* or *order ID* cannot be changed.

To implement a field action that disables field input, follow these instructions:

1. In the Visual Studio, create a new project.
2. Add references to the `references to Sitecore.Kernel.dll` and `Sitecore.Forms.Core.dll` assemblies.
3. Create a new class that inherits `Sitecore.Rules.Actions.RuleAction<T> where T : Sitecore.Forms.Core.Rules.ConditionalRuleContext.`
4. Override and implement the `Apply(T ruleContext)` method. This method is responsible for executing any actions.
5. `ruleContext.Control` contains the field or section that is selected in the Form Designer:

```
namespace Sitecore.Forms.Core.Rules
{
  using System.Web.UI.WebControls;
  using Sitecore.Diagnostics;
  using Sitecore.Rules.Actions;

  /// <summary>
  /// The disable control action
  /// </summary>
  /// <typeparam name="T"></typeparam>
  public class DisableControlAction<T> : RuleAction<T> where T : ConditionalRuleContext
  {
    /// <summary>
    /// Applies the specified rule context.
    /// </summary>
    /// <param name="ruleContext">The rule context.</param>
    public override void Apply(T ruleContext)
    {
      Assert.ArgumentNotNull(ruleContext, "ruleContext");

      if (ruleContext.Control != null && ruleContext.Control is WebControl)
      {
        ((WebControl)ruleContext.Control).Enabled = false;
      }
    }
  }
}
```

6. In the Content Editor, in the `/sitecore/system/Settings/Rules/Web Forms for Marketers Conditions/Actions` folder create a new action item using the action template



Disable Element

Quick Info

Item ID: {BCDA6E12-A967-4D2F-B080-D6D09B1E5070}
Item Name: Disable Element
Item Path: /sitecore/system/Settings/Rules/Web Forms for Marketers Conditions/Actions/Disable Element
Template: /sitecore/templates/System/Rules/Action - {F90052A5-B4E6-4E6D-9812-1E1B88A6FCEA}
Created From: [unknown]
Item Owner: sitecore\admin

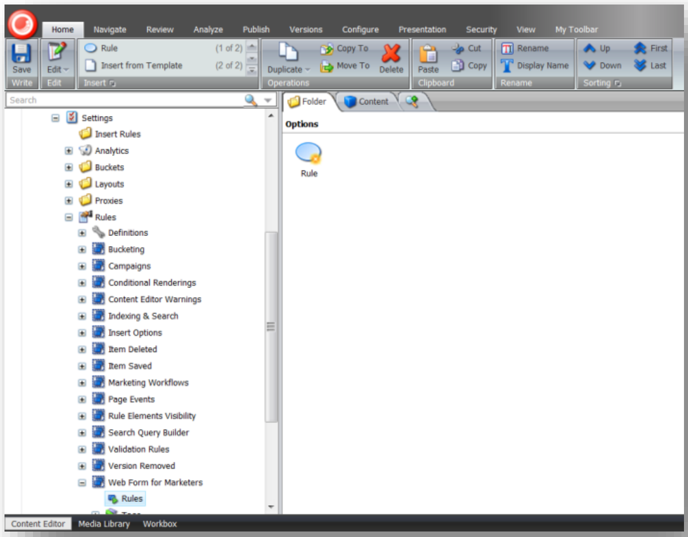
Data

Text:
disable element

Script

Type [shared]:
Sitecore.Forms.Core.Rules.DisableControlAction,MyAssembly

Code [shared]:

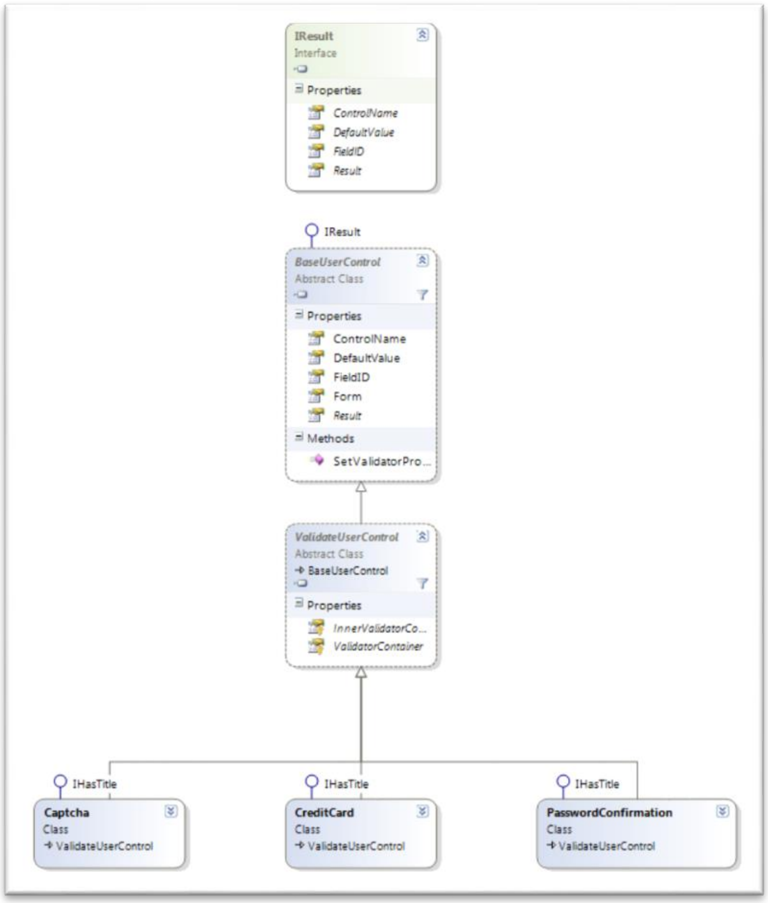


7. Fill in the text and type fields.

3.7 What Field Hierarchy is used in the Module?

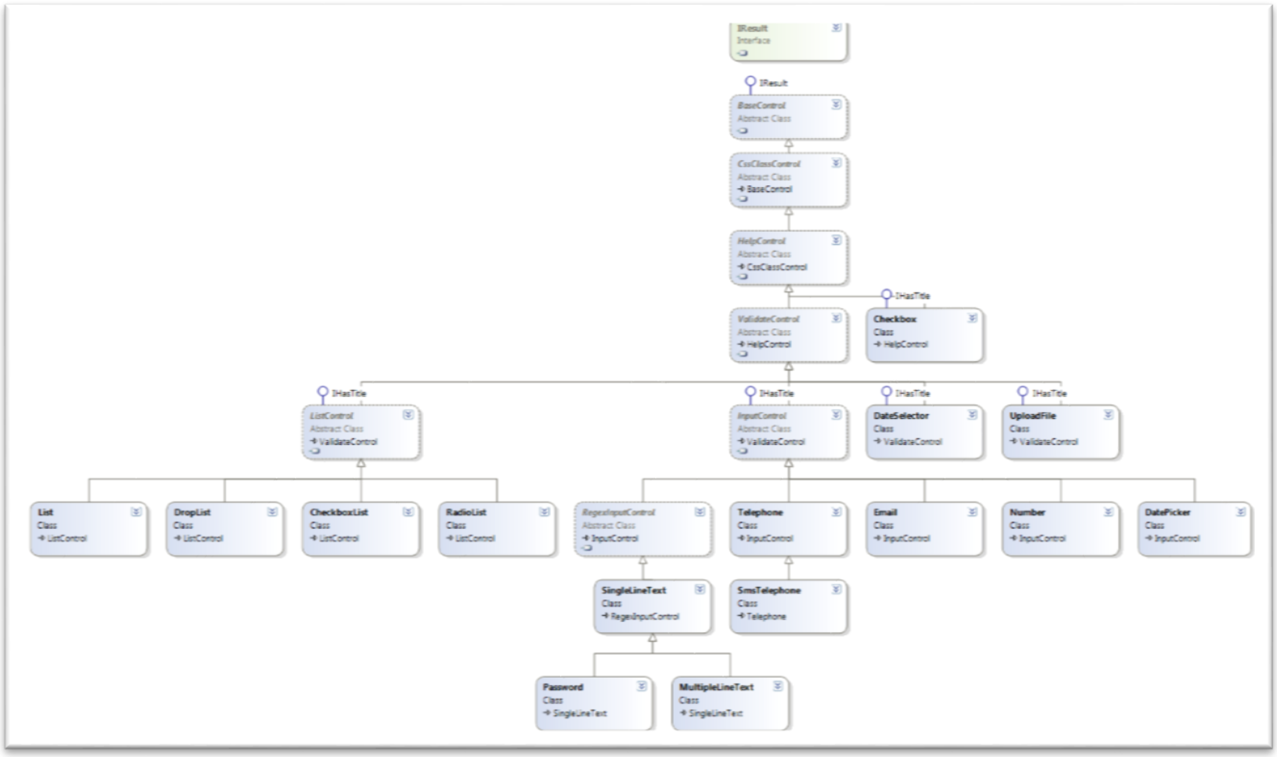
There are two groups of fields in the Web Forms for Marketers module:

- Fields that inherit the `System.Web.UI.UserControl` class. The markup of these fields can be used in a separate `.ascx` file. You can change the appearance of these fields without recompiling assemblies. There are three fields of this group in the `Sitecore.Form.UI.UserControls` namespace:
  - Captcha
  - CreditCard
  - PasswordConfirmation



- Fields that inherit the `System.Web.UI.WebControls.WebControl` class. Usually, these fields have a simple structure that does not require any modifications after compiling the assemblies. Fields of this group are easier to reuse than the fields that inherit the `System.Web.UI.UserControl` class. These fields are located in the `Sitecore.Form.Web.UI.Controls` namespace:
  - Checkbox
  - CheckboxList

- DatePicker
- DateSelector
- DropList
- Email
- Label
- List
- MultipleLineText
- Number
- Password
- RadioList
- SingleLineText
- SmsTelephone
- Telephone
- UploadFile



3.8 How to Create a New Field Type

To create a new field type in the Web Forms for Marketers module follow these instructions:

1. In the Visual Studio, create a new web application project for the existing Sitecore solution.

   For more information about creating new project, see the section *How to Create a Visual Studio Web Application Project* in the following document:
   http://sdn.sitecore.net/upload/sitecore6/64/presentation_component_cookbook-a4.pdf

2. Add a reference to the `Sitecore.Forms.Core` assembly.

3. In your project, add an item basing on the **Class** template.

4. The module has two groups of fields. Analyze which group the new field relates to. For more information about field hierarchy, see the section *What Field Hierarchy is used in the Module?* Inherit one of the following group of classes:

   o The `Sitecore.Form.Web.UI.Controls.IResult` and `System.Web.UI.WebControls.WebControl` class. Alternatively, inherit the `Sitecore.Form.Web.UI.Controls.ValidateControl` class that already contains needed parameters of both classes.

   o The `Sitecore.Form.Web.UI.Controls.IResult` and `System.Web.UI.UserControl` class. Alternatively, inherit the `Sitecore.Form.Web.UI.Controls.ValidateUserControl` class that already contains needed parameters of both classes. A field of this group is based on the `.ascx` file.

   The `Sitecore.Form.Web.UI.Controls.IResult` interface lets the module to read the field values.

```
   public interface IResult
   {
      ControlResult Result { get; set; }

      string ControlName { get; set; }

      string FieldID { get; set; }

      string DefaultValue { set; }
   }
```

   `Result` — returns the field value of the object in the `ControlResult` class.

   `ControlName` — get or set the system item name of the field type.

   `FieldID` — the ID of the item of the field type.

   `DefaultValue` – default field value.

   If the returned value is not a string, you must set an adapter for the *Result* value. To set an adapter, use the `Adapter` attribute before the class definition:

```
namespace Sitecore.Form.Web.UI.Controls
{
   [Adapter(typeof (DateAdapter))]
   [ValidationProperty("Value")]
   public class CustomField : ValidateControl
```

   The type of this attribute must inherit the `Sitecore.Form.Core.Client.Submit.Adapter` abstract class.

```
   public class DateAdapter : Sitecore.Form.Core.Client.Submit.Adapter
   {
            public DateAdapter () {  }
      // Convert object value to string
public override string AdaptResult(object value)
      {
         return value.ToString();
      }
// Convert value from database to friendly value. Using in the Form Data Viewer
public override string AdaptToFrendlyValue(string value)
      {
         return value;
      }
// Convert list value from database to friendly value. Using in the Form Data
Viewer
      public override IEnumerable<string> AdaptToFrendlyListValues(string value)
      {
         return new List<string>(new[]{value});
      }
   }
```

5. If you want to let the user enter the field title in the Form Designer, implement the `Sitecore.Form.Web.UI.Controls.IHasTitle` interface:

```
   public interface IHasTitle
   {
      string Title { set; get; }
   }
```

It contains the only *Title* property. The module uses this property to set the field title.

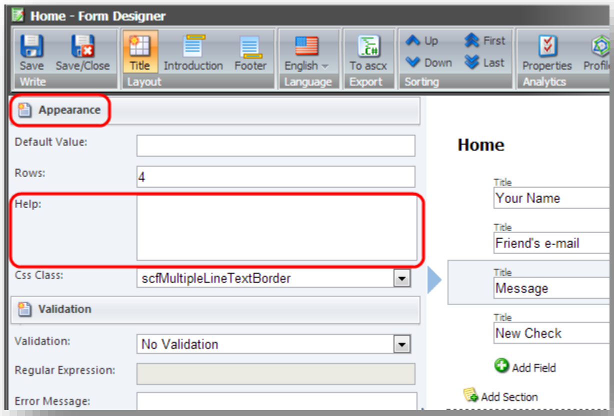
6. If you want to make the properties of your control available to the user in the Form Designer, use the following attributes:

   - *VisualProperty* — displays the property in the Form Designer.
   - *VisualCategory* — sets the category name.
   - *VisualFieldType* — defines the type of the input control for the property. If the attribute is not specified, the property has the EditField input type. You can find the list of available visible field types under the Sitecore.Form.Core.Visual namespace.
   - *DefaultValue* — sets the deault value in the Form Designer.
   - *Localize* — defines whether the property supports multiple languages or not.

   For example, to make the *Help* property available in the Form Designer, in the **Appearance** section with the 500 sort order, you can use this code:

```
[VisualProperty("Help:", 500)]
[VisualCategory("Appearance")]
[VisualFieldType(typeof(TextAreaField)), Localize]
public string Information
{
  ...
}
```

The Help field is now displayed:



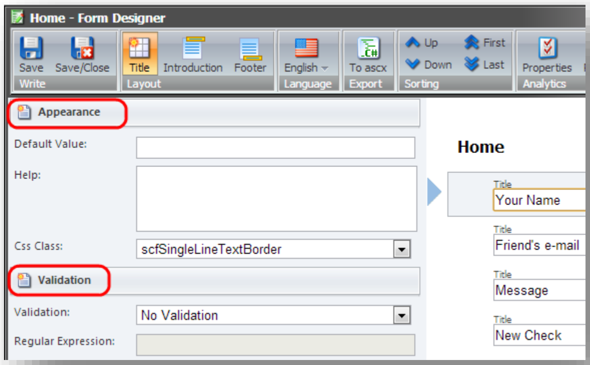
Default Value:
Rows: 4
Help:
Css Class: scfMultipleLineTextBorder
Validation: No Validation
Regular Expression:
Error Message:

Home
Title: Your Name
Title: Friend's e-mail
Title: Message
Title: New Check

7. Build the project and put the compiled DLL file in the *\bin* folder of your Sitecore solution.
8. In the **Content Tree**, in the `/sitecore/system/Modules/Web Forms for Marketers/Settings/Field Types/Custom` folder, create an item on the `/sitecore/templates/Web Forms for Marketers/Field` Type template.
   1) If your field type inherits the `System.Web.UI.UserControl` class, in the field type item, in the **User Control** field, enter the path to the corresponding `ascx` file.
   2) If your field type inherits the `System.Web.UI.WebControls.WebControl` class, in the field type item, in the **Assembly** field enter the name of the DLL file and in the **Class** field, enter the name of the corresponding class.

3.8.1 Creating a Visual Field Property

In this section, we describe how to create a property section for a field type that is displayed in the Form Designer:



Default Value:
Help:
Css Class: scfSingleLineTextBorder
Validation: No Validation
Regular Expression:

Home
Title: Your Name
Title: Friend's e-mail
Title: Message
Title: New Check

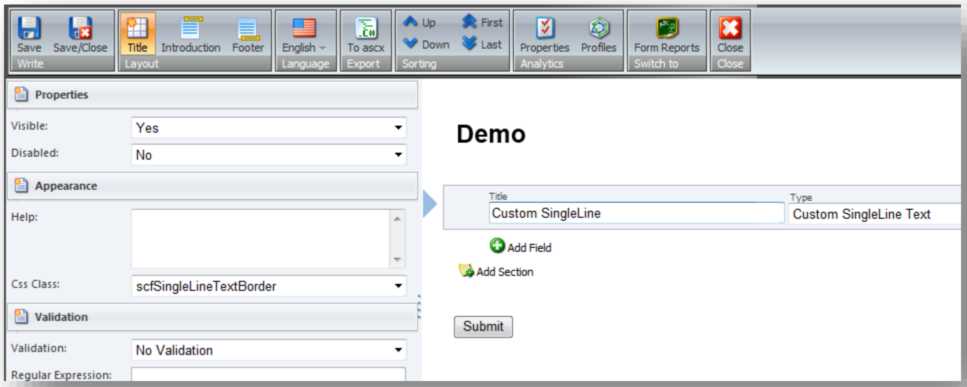
In this example, we create a new field type that is inherited from the `Sitecore.Form.Web.UI.Controls.SingleLineText` class and has two properties: *Disabled* and *Visible*.

1. In the Visual Studio, create a new project and name it, for example, *CustomSingleLineText*.
2. Add new references to the `Sitecore.Forms.Core` and the `Sitecore.Forms.Custom` assemblies.
3. Create a new class that inherits `Sitecore.Form.Web.UI.Controls.SingleLineText` class and add the *Disabled* and the *Visible* visual properties:

```
        namespace Sitecore.Custom.Form.Web.UI.Controls
        {
        public class CustomSingleLineText : SingleLineText
          {
            // Set field enabled or disabled
            [DefaultValue("No"), VisualFieldType(typeof(BooleanField)),
VisualProperty("Disabled:", 100), VisualCategory("Properties")]
            public string IsDisabled
            {
              get
              {
                return base.textbox.Enabled.ToString();
              }
              set
              {
                base.textbox.Enabled = value == "No";
              }
            }

            //// Set field visible or not
            [DefaultValue("Yes"), VisualFieldType(typeof(BooleanField)),
VisualProperty("Visible:", 100), VisualCategory("Properties")]
            public string IsVisible
            {
              get
              {
                return base.textbox.ToString();
              }
              set
              {
                base.textbox.Visible = value == "Yes";
                base.title.Visible = value == "Yes";

              }
            }
          }
        }
```

4. Build the project and put the compiled `CustomSingleLineText.dll` file in the *\website\bin\* folder.
5. In the **Content Editor**, in the */sitecore/system/Modules/Web Forms for Marketers/Settings/Field Types/Custom* folder, create a new **Custom SingleLineText** item based on the */sitecore/templates/Web Forms for Marketers/Field Type* template.
6. In the **Custom SingleLineText** item, in the **Assembly** field, enter *CustomSingleLineText* and in the **Class** field enter *Sitecore.Custom.Form.Web.UI.Controls.CustomSingleLineText*.

Here is how the new field type and its properties are rendered in the Form Designer:



Properties

Visible: Yes

Disabled: No

Appearance

Help:

Css Class: scfSingleLineTextBorder

Validation

Validation: No Validation

Regular Expression:

Demo

Title	Type
Custom SingleLine	Custom SingleLine Text

3.8.2 Creating a New Field Type Based on the WebControl Class

In this section, we describe how to create a new field type based on the *WebControl* class by giving an example of the *DatePicker* field.

To create a *DatePicker* field type:

1. In the Visual Studio, create a new project and name it, for example, *Demo.Examples*.
2. Add a new reference to the `Sitecore.Forms.Core` assembly.
3. Create a new class that inherits `Sitecore.Form.Web.UI.Controls.ValidateControl`:

```
namespace Demo.Examples
{
  using System;
  using System.Web.UI;
  using System.Web.UI.WebControls;
  using Sitecore.Form.Core.Attributes;
  using Sitecore.Form.Core.Controls.Data;
  using Sitecore.Form.Core.Visual;
  using Sitecore.Form.Web.UI.Controls;

  /// <summary>
  ///   The date picker control
  /// </summary>
  public class DatePicker : ValidateControl, IHasTitle
  {
    /// Here we define the controls that make the structure of the field
    /// <summary>
    /// The control container
    /// </summary>
    protected Panel container = new Panel();

    /// <summary>
    ///  The help notes of the control
    /// </summary>
    protected Label helpNotes = new Label();

    /// <summary>
    ///  The input for the control
    /// </summary>
    protected TextBox input = new TextBox();

    /// <summary>
```

```
            ///  The title of the control
            /// </summary>
            protected Label title = new Label();

            ///the constructor that defines the default values of the field properties
            /// <summary>
            /// Initializes a new instance of the <see cref="DatePicker"/> class.
            /// </summary>
            public DatePicker()
              : base(HtmlTextWriterTag.Div)
            {
              this.CssClass = "scfDatePickerBorder";

            }

            /// Implementation of the ihastitle interface to get the title that is entered in the
Form Designer
            /// <summary>
            /// Gets or sets the title of the control.
            /// </summary>
            /// <value>The title.</value>
            public string Title
            {
              get
              {
                return this.title.Text;
              }

              set
              {
                this.title.Text = value;
              }
            }
            /// Implementation of the property that is set in the Form Designer
            /// <summary>
            /// Gets or sets the help notes for the control.
            /// </summary>
            /// <value>The text for help notes.</value>
            [VisualProperty("Help:", 500), VisualCategory("Appearance"),
VisualFieldType(typeof(TextAreaField)), Localize]
            public string HelpNotes
            {
              get
              {
                return this.helpNotes.Text;
              }

              set
              {
                this.helpNotes.Text = value ?? string.Empty;
              }
            }
            ///The result of the control that will be used in the save actions and verification
actions later
            /// <summary>
            /// Gets the result of clients' inputs.
            /// </summary>
            /// <value></value>
            public override ControlResult Result
            {
              get
              {
                return new ControlResult(this.ControlName, this.textbox.Text, null);
              }

              set
              {
                this.textbox.Text = value.Value.ToString();
              }
            }

            ///Creating the structure of the field
            /// <summary>
            /// Builds the control structure
            /// </summary>
            /// <param name="e">An <see cref="T:System.EventArgs"/> object that contains the
event data.</param>
            protected override void OnInit(EventArgs e)
            {
              this.input.CssClass = "scfDatePickerTextBox";
              this.helpNotes.CssClass = "scfDatePickerUsefulInfo";
              this.container.CssClass = "scfDatePickerGeneralPanel";
              this.title.CssClass = "scfDatePickerLabel";

              this.input.TextMode = TextBoxMode.SingleLine;

              this.Controls.AddAt(0, this.container);
              this.Controls.AddAt(0, this.title);

              this.container.Controls.AddAt(0, this.helpNotes);
              this.container.Controls.AddAt(0, this.input);
            }

            /// <summary>
            /// Attaches jscripts to the page
            /// </summary>
            /// <param name="e">The e.</param>
            protected override void OnPreRender(EventArgs e)
            {
              string script = "$(document).ready(function() { $('#" + this.input.ClientID +
"').datepicker()";

              this.Page.ClientScript.RegisterClientScriptInclude("jquery", "/sitecore
modules/web/web forms for marketers/scripts/jquery.js");
              this.Page.ClientScript.RegisterClientScriptInclude("jquery.ui", "/sitecore
modules/web/web forms for marketers/scripts/jquery-ui.min.js");
              this.Page.ClientScript.RegisterStartupScript(this.GetType(), script, script, true);

              base.OnPreRender(e);
            }
          }
        }
```

4. Build the project and put the compiled `Custom DatePicker.dll` file in the `\website\bin\` folder.
5. In the **Content Editor**, in the `/sitecore/system/Modules/Web Forms for Marketers/Settings/Field Types/Custom` folder, add a new **Custom DatePicker** item using `/sitecore/templates/Web Forms for Marketers/Field Type` template.
6. In the **Custom DatePicker** item, in the **Assembly** field, enter *Custom DatePicker* and in the **Class** field enter *DatePicker*.

3.8.3 Creating a New Field Type Based on the UserControl Class - .ascx file

To create a new field type based on the UserControl class (.ascx file):

1. In Visual Studio, create a new project and name it, for example, *UserControlField*.
2. Add a new reference to the `Sitecore.Forms.Core` assembly.

3. Create a new class that inherits `Sitecore.Form.Web.UI.Controls.ValidateUserControl`. To give a new field a title, inherit your class from the `IHasTitle` interface:

```
public class CreditCard : ValidateUserControl, IHasTitle
{
….
}
```

4. Create the *UserControlField.ascx* user control file in the *\Website\sitecore modules\Web\Web Forms for Marketers\UI\UserControl* folder.
5. Build the project and put the compiled *Custom UserControlField.dll* file in the *\website\bin\* folder.
6. In the **Content Editor**, in the */sitecore/system/Modules/Web Forms for Marketers/Settings/Field Types/Custom* folder, create a new **Custom UserControlField** item based on the */sitecore/templates/Web Forms for Marketers/Field Type* template.
7. In the **Custom UserControlField** item, in the **User Control** field, enter the path to the corresponding ascx file: */sitecore modules/web/Web Forms for Marketers/UI/UserControl/ UserControlField.ascx*.

3.9 Creating a Save Action

You can create a custom save action. Use the base interface and classes when you are creating a new class.

3.9.1 Base Interface and Classes

To create a custom save action, you must create a class that inherits the `Sitecore.Form.Submit.ISaveAction` interface. This interface contains the Execute method that must be implemented in your new class, because it is called by all save actions assigned to a web form. Use this method to implement a save action logic.

The Execute method accepts three arguments:

- *ID formid* – id of the web form item that the action is assigned to.
- *AdaptedResultList* fields – a list of the *AdaptedResult* classes. Each item in the list provides information about a web form field that is submitted (*Value*, *FieldID*, *FieldName*, *Parameters*).
- *params object[]* data – the first element of the array that contains the analytics session ID.

The Web Forms for Marketers module provides several classes that you can inherit a custom class from:

- `Sitecore.Form.Submit.UserBaseAction` class
- `Sitecore.Form.Core.Submit.AuditSaveAction` class

Sitecore.Form.Submit.UserBaseAction class

The `Sitecore.Form.Submit.UserBaseAction` class lets you manage users' accounts.

The `Sitecore.Form.Submit.UserBaseAction` class contains the following methods:

Method	**Description**
formUploadFile	Runs when a file is uploaded using the FileUpload field of a web form.
successAction	Runs when a web form is submitted successfully. Default processors redirect the user to the success page or generate the success message.
exportToExcel	Runs when the module exports web forms data to a Microsoft Excel document from the Data Viewer.
parseAscx	Runs when the module launches the **Convert to ascx** dialog box. This pipeline parses a web form and generates the output .ascx control.
exportToAscx	Runs when the user clicks **Download** in the **Convert to ascx** dialog box. This pipeline generates an .ascx control for downloading.
auditRender	Writes audit information to the user profile when the module performs a save action related to security, for example the Create User save action.
processMessage	Runs when the module generates an email message for the Send Email Message save action.

3.9.2 Commands

This section contains UI commands specific to the Web Forms for Marketers module.

For example, when the user in the Content Editor, on the Presentation tab, clicks the **Insert** button the `forms:insert` command is performed.

3.9.3 Settings

This section contains the settings that you can use to configure the Web Forms for Marketers module. Each setting contains a short description that will help to understand how this setting works.

3.10 How to Create an Action Editor

Action Editor is a dialog box that lets the user set some parameters of a save action. You can create an Action Editor for your custom save action.

Before creating an Action Editor, refer to the following articles for more information about xml controls:

- http://sdn.sitecore.net/Articles/XML%20Sheer%20UI/Beginning%20with%20XML%20controls.aspx
- http://sdn.sitecore.net/SDN5/Articles/XML%20Sheer%20UI/My%20first%20XML%20application.aspx

The working logic of the Action Editor is the following:

- Read save action parameters in the `OnLoad` method.
- Show save action parameters to the user.
- After the user edited the parameters, save them to the **Save Actions** field of the web form using the `OnOk` method.

Analyze the following code tips before implementing an Action Editor:

- To get current save action parameters:

```
        string params=HttpContext.Current.Session[Sitecore.Web.WebUtil.GetQueryString("params")]
as string;
        NameValueCollection nvParams=ParametersUtil.XmlToNameValueCollection(params);
```

- To save parameter values, override the `OnOk` method:

```
        protected override void OnOK(object sender, EventArgs args)
            {
              string str3 = ParametersUtil.NameValueCollectionToXml(this.nvParams ?? new
NameValueCollection());
              if (str3.Length == 0)
              {
                str3 = "-";
              }
              SheerResponse.SetDialogValue(str3);
              base.OnOK(sender, args);
            }
```

- To get the current web form item ID:

```
        Sitecore.Web.WebUtil.GetQueryString("id");
```

- To get the current language:

```
        Sitecore.Web.WebUtil.GetQueryString("la", "en");
```

- To get the current Sitecore database:

```
        Sitecore.Web.WebUtil.GetQueryString("db");
```

Perform the following actions to create an action editor:

1. Create a `*xml` file that contains the dialog layout:

```
        <?xml version="1.0" encoding="utf-8" ?>
        <control xmlns:def="Definition" xmlns="http://schemas.sitecore.net/Visual-Studio-
Intellisense">
           <SimpleEditor>
              <Stylesheet>
                .scfContent {
```

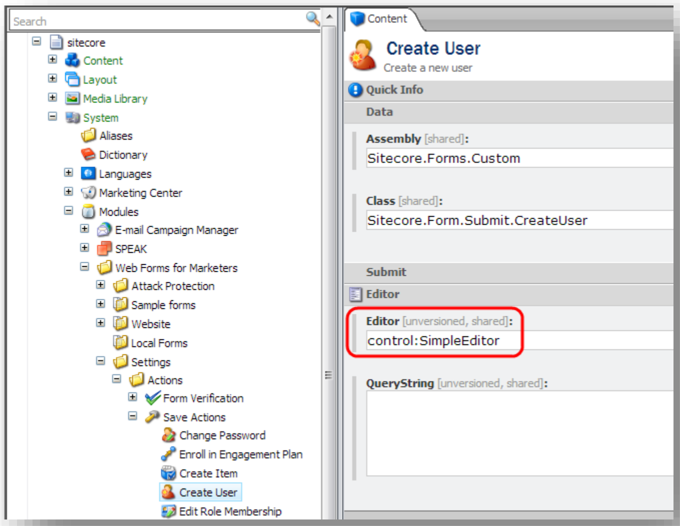
```
                padding-top : 15px;
                }

                .scfFieldScope{
                width:100%; margin:7px;
                }

                .scfFieldLabel {
                width:40%;
                }

                .scfFieldSelect {
                width:58%;
                position:absolute;
                right:0;
                margin-right:20px;
                }
              </Stylesheet>

              <FormDialog ID="Dialog" Icon="Software/32x32/step_new.png">
                 <CodeBeside Type="Sitecore.Forms.Sample.SimpleEditor,Sitecore.Forms.Sample"/>
                <Border Class="scfContent" Width="100%"  Align="left" Style="overflow:none;">
                  <Literal ID="name" Text="Login:" Class="scfFieldLabel"/>
                  <Combobox runat="server" ID="login"/>
                </Border>
                <Border Class="scfContent" Width="100%"  Align="left" Style="overflow:none;">
                  <Literal ID="pass" Text="Password:" Class="scfFieldLabel"/>
                  <Combobox runat="server" ID="password"/>
                </Border>
                 <Border Class="scfContent" Width="100%"  Align="left" Style="overflow:none;">
                   <Literal ID="domLiteral" Text="Default domain:" Class="scfFieldLabel"/>
                   <Combobox runat="server" ID="domain"/>
                 </Border>
              </FormDialog>
           </SimpleEditor>
        </control>
```

2. Create an action editor class based on the `Sitecore.Web.UI.Pages.DialogForm` one. Specify the created class as `CodeBeside` in the `*.xml` file. Here is the action editor class sample code:

```
        public class SimpleEditor : DialogForm
        {
            protected Combobox domain;
            protected Combobox login;
            protected Combobox password;
            private NameValueCollection nvParams;
            protected override void OnLoad(EventArgs e)
            {
              base.OnLoad(e);
              if (!Context.ClientPage.IsEvent)
              {
                foreach (Domain d in DomainManager.GetDomains())
                {
                  Sitecore.Web.UI.HtmlControls.ListItem item = new
Web.UI.HtmlControls.ListItem();
                  item.Value = d.Name;
                  item.Header = d.Name;
                  domain.Controls.Add(item);
                }
                foreach (FieldItem f in this.CurrentForm.Fields)
                {
                  Sitecore.Web.UI.HtmlControls.ListItem item = new
Web.UI.HtmlControls.ListItem();
                  item.Value = f.ID.ToString();
                  item.Header = f.DisplayName;
                  login.Controls.Add(item);
                  item = new Web.UI.HtmlControls.ListItem();
                  item.Value = f.ID.ToString();
                  item.Header = f.DisplayName;
                  password.Controls.Add(item);
                }
                domain.Value = this.GetValueByKey("DefaultDomain") ?? string.Empty;
                login.Value = this.GetValueByKey("Login") ?? string.Empty;
                password.Value=this.GetValueByKey("Password") ??string.Empty;
              }
            }
            protected void SaveValues()
            {
              this.SetValue("DefaultDomain", domain.Value);
              this.SetValue("Login", login.Value);
              this.SetValue("Password", password.Value);
            }
            protected override void OnOK(object sender, EventArgs args)
            {
              this.SaveValues();
              string str3 = ParametersUtil.NameValueCollectionToXml(this.nvParams ?? new
NameValueCollection());
              if (str3.Length == 0)
              {
                str3 = "-";
              }
              SheerResponse.SetDialogValue(str3);
              base.OnOK(sender, args);
            }

            public void SetValue(string key, string value)
            {
              if (this.nvParams == null)
              {
                this.nvParams = ParametersUtil.XmlToNameValueCollection(this.Params);
              }
              this.nvParams[key] = value;
            }

            public string GetValueByKey(string key)
            {
              if (this.nvParams == null)
              {
                this.nvParams = ParametersUtil.XmlToNameValueCollection(this.Params);
              }
              return (this.nvParams[key] ?? string.Empty);
            }
            public string Params
            {
              get
              {
                return
(HttpContext.Current.Session[Sitecore.Web.WebUtil.GetQueryString("params")] as string);
              }
            }

            public string CurrentID
            {
              get
              {
                return Sitecore.Web.WebUtil.GetQueryString("id");
              }
            }
            public FormItem CurrentForm
            {
              get
              {
                if (!string.IsNullOrEmpty(this.CurrentID))
                {
                  Item innerItem = this.CurrentDatabase.GetItem(this.CurrentID,
this.CurrentLanguage);
```

```
        if (innerItem != null)
        {
          return new FormItem(innerItem);
        }
      }
      return null;
    }
  }
  public virtual Database CurrentDatabase
  {
    get
    {
      return Factory.GetDatabase(Sitecore.Web.WebUtil.GetQueryString("db"));
    }
  }
  public virtual Language CurrentLanguage
  {
    get
    {
      return Language.Parse(Sitecore.Web.WebUtil.GetQueryString("la", "en"));
    }
  }
}
```

3. Specify the created editor in the appropriate Save Action item, in the **Editor** field:



Create User
Create a new user

Assembly [shared]: Sitecore.Forms.Custom

Class [shared]: Sitecore.Form.Submit.CreateUser

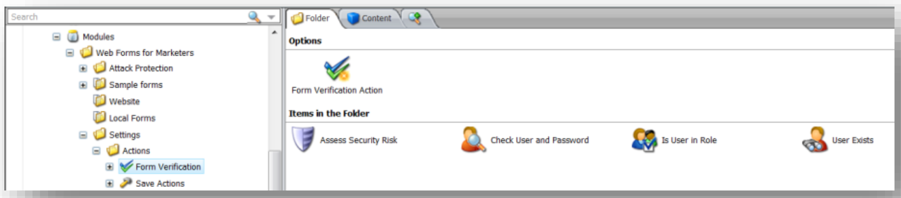
Editor [unversioned, shared]: control:SimpleEditor

QueryString [unversioned, shared]:

3.11 How to Create a Form Verification Action

To create a form verification action:

1. In the **Content Editor**, select the `sitecore/content/system/modules/web forms for marketers/settings/actions/form verification` item.
2. Create a new form verification action item and add the appropriate parameters.

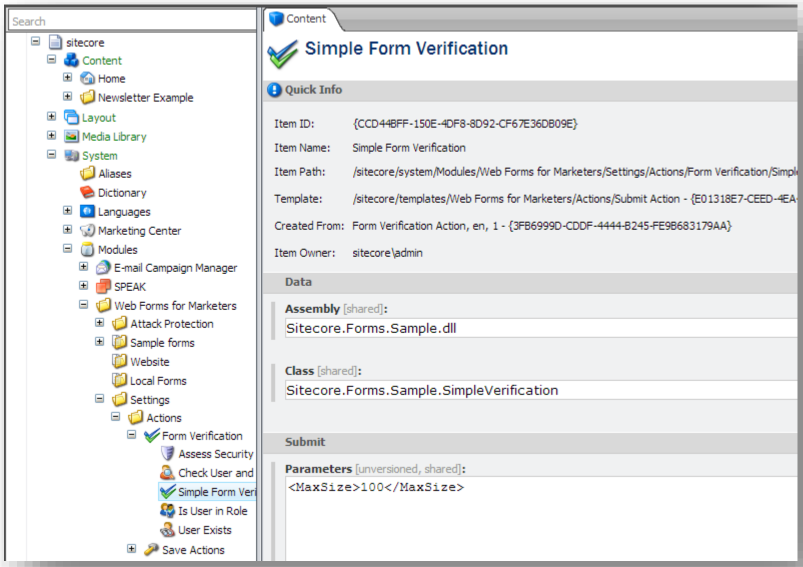


Creating a new form verification action is similar to the creating new save action. For more information about creating a save action, see the *Creating a Save Action* section.

The only difference is that you must use the `Sitecore.Form.Core.Submit.BaseCheckAction` class as a base class for your verification action. To indicate that verification is failed, the module must throw an exception generated by the actions code. Module handles it and displays an exception message to the user. The following example shows a verification action that checks whether a size of files uploaded using the **UploadFile** field is less than the limited one:

```
        namespace Sitecore.Forms.Sample
        {
          class SimpleVerification:BaseCheckAction
          {
            public override void Execute(Sitecore.Data.ID formid,
IEnumerable<Form.Core.Controls.Data.ControlResult> fields)
            {
              foreach (ControlResult cr in fields)
              {
                PostedFile file= cr.Value as PostedFile;
                if(file!=null)
                {
                  if (file.Data.Length > int.Parse(MaxSize))
                  {
                    throw new Exception("File size should be less than " + MaxSize+" bytes");
                  }
                }
              }
            }
            public string MaxSize { get; set; }
          }
        }
```

In our example, the action definition item may look like this:



Simple Form Verification

Quick Info

Item ID:	{CCD44BFF-150E-4DF8-8D92-CF67E36DB09E}
Item Name:	Simple Form Verification
Item Path:	/sitecore/system/Modules/Web Forms for Marketers/Settings/Actions/Form Verification/Simpl
Template:	/sitecore/templates/Web Forms for Marketers/Actions/Submit Action - {E01318E7-CEED-4EA
Created From:	Form Verification Action, en, 1 - {3FB6999D-CDDF-4444-B245-FE9B683179AA}
Item Owner:	sitecore\admin

Data

Assembly [shared]:
Sitecore.Forms.Sample.dll

Class [shared]:
Sitecore.Forms.Sample.SimpleVerification

Submit

Parameters [unversioned, shared]:
```
<MaxSize>100</MaxSize>
```

3.12 How to Access Submitted Web Form Data

If you assigned the Save to Custom Database save action to a web form, all the data entered by a website visitor is saved to the user custom database. You can read the saved data programmatically.

In the example below, we will perform the following actions:

- Get all the submits of a particular web form;
- Get the value of the **Age** field for each web form;

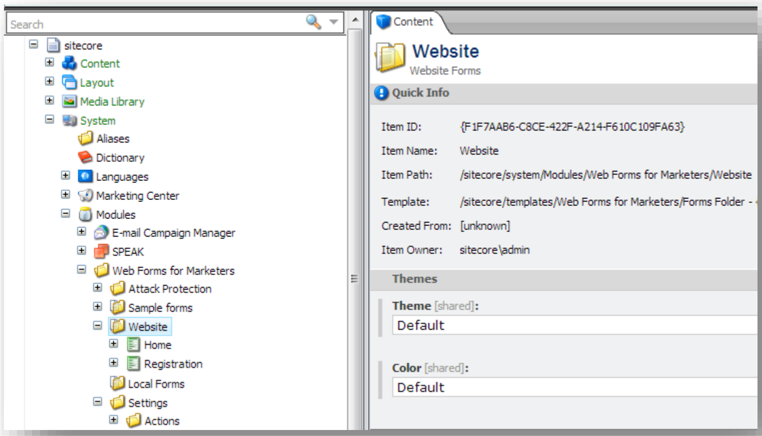
```
        string formId = "{C797CBE2-B5B9-4C5B-9B60-50438A1783A8}"; // replace the value with ID of
your form item
        List<GridFilter> args = new List<Sitecore.Web.UI.Grids.GridFilter>();
        args.Add(new GridFilter("storageName", string.Empty,
GridFilter.FilterOperator.Contains));
        args.Add(new GridFilter("dataKey", formId, GridFilter.FilterOperator.Contains));
        var submits = FormDataManager.Instance.GetForms().GetPage(new PageCriteria(0,
0x7ffffffe), null, args);

        foreach (IForm submit in submits)
        {
          foreach (IField field in submit.Field)
          {
            if (field.FieldName == "Age")
            {
              var fieldValue= field.Value;
              // process field value logic
            }
          }
        }
```

3.13 How to Use CSS Themes

The module contains a few CSS themes that you can apply to web forms. To change a theme, in the Content Editor, navigate to the folder in which the web forms are stored. This folder is defined in the `Sitecore.Forms.config` file, in the `formsRoot` parameter for the website. This folder is based on the `/sitecore/Templates/Web Forms for Marketers/Forms Folder` template.

The web forms folder contains the following fields:

- **Theme** — sets the appearance theme that is used for all the forms.
- **Color** — sets the color theme that is used for all the forms.



Website
Website Forms

Quick Info

Item ID: {F1F7AAB6-C8CE-422F-A214-F610C109FA63}
Item Name: Website
Item Path: /sitecore/system/Modules/Web Forms for Marketers/Website
Template: /sitecore/templates/Web Forms for Marketers/Forms Folder -
Created From: [unknown]
Item Owner: sitecore\admin

Themes

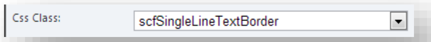
Theme [shared]: Default

Color [shared]: Default

All the web forms for the current website have the same theme and color, because these parameters set for the web forms folder.

If you want to extend the list of available themes, you must register the new theme. You must create a new item under the `/sitecore/System/Modules/Web Forms for Marketers/Settings/Meta data/Themes` folder or the `/sitecore/System/Modules/Web Forms for Marketers/Settings/Meta data/Colors` folder. The name of this item must coincide with the name of the file where you defined the CSS styles. This file must be in the `sitecore modules/shell/web forms for marketers/themes` or `sitecore modules/shell/web forms for marketers/themes/colors` folder.

You can also change the CSS class at field level. In the **Form Designer**, you can use `Css Class` property that is available for every field.



Css Class: scfSingleLineTextBorder

To extend or add a new CSS classes to the list:

1. Add the definition of the CSS class to the website\`sitecore modules\Shell\Web Forms for Marketers\Themes\Custom.css` file.

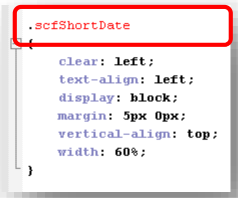
2. Under the `/sitecore/System/Modules/Web Forms for Marketers/Settings/Meta data/Css Classes` folder, create a new item based on the `/sitecore/Templates/Web Forms for Marketers/Meta Data/Extended List Item` template.
3. In the **Value** field of the new item, add the name of your CSS class.

3.14 How to Configure CSS Styles

The module contains a few CSS styles that you can apply to form fields. Every field type has a default CCS style that is applied to it. You can customize CSS styles.

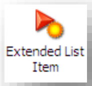
To add a new CSS style:

1. In the `custom.css` file, located under the `Website\sitecore modules\Shell\Web Forms for Marketers\Themes\` folder, define a new CSS style.

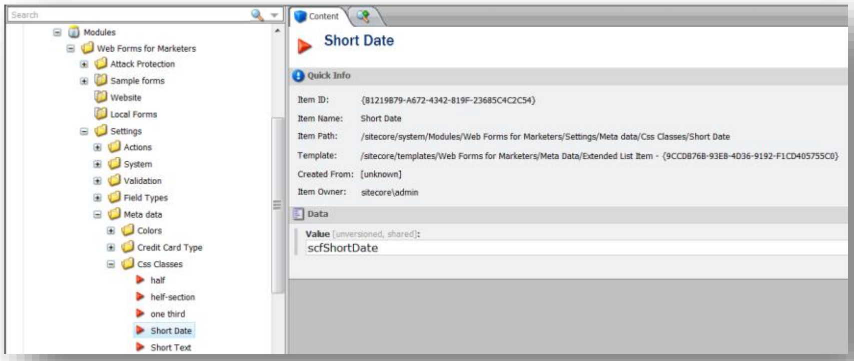


```
.scfShortDate
{
    clear: left;
    text-align: left;
    display: block;
    margin: 5px 0px;
    vertical-align: top;
    width: 60%;
}
```

2. In the **Content Editor**, navigate to the `Sitecore/System/Modules/Web Forms for Marketers/Settings/Meta data/CSS Classes` folder.
3. Create a new **Extended List item**.



4. In the new item, in the value field, enter the name of the CSS style.



Short Date

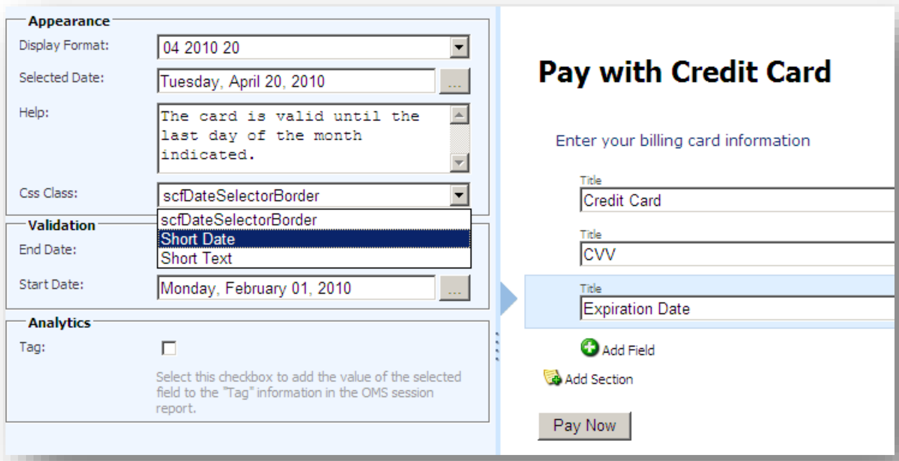
Quick Info

Item ID: {B1219B79-A672-4342-819F-23685C4C2C54}
Item Name: Short Date
Item Path: /sitecore/system/Modules/Web Forms for Marketers/Settings/Meta data/Css Classes/Short Date
Template: /sitecore/templates/Web Forms for Marketers/Meta Data/Extended List Item - {9CCDB76B-93E8-4D36-9192-F1CD405755C0}
Created From: [unknown]
Item Owner: sitecore\admin

Data

Value [unversioned, shared]:
scfShortDate

Your custom CSS style is added to the system. You can apply it to any form field in the **Form Designer**:



Appearance
Display Format: 04 2010 20
Selected Date: Tuesday, April 20, 2010
Help: The card is valid until the last day of the month indicated.
Css Class: scfDateSelectorBorder
scfDateSelectorBorder
Short Date
Short Text
Validation
End Date:
Start Date: Monday, February 01, 2010
Analytics
Tag:
Select this checkbox to add the value of the selected field to the "Tag" information in the OMS session report.

Pay with Credit Card

Enter your billing card information

Title
Credit Card
Title
CVV
Title
Expiration Date

3.15 How to Configure the Web Forms for Marketers Module to work with MVC

To enable web forms work on MVC, you must create an MVC layout for the item in which you want to place the MVC Form rendering. For more information about how to enable Sitecore MVC, see the Sitecore MVC Playground article.

3.15.1 How the MVC version is connected to the WebForms version

The old items structure was reused for MVC implementation, except `/sitecore/system/Modules/Web Forms for Marketers/Settings/Validation`.

Save actions, saving data to analytics still work as in the `WebForms` version.

3.15.2 How to Start Using the MVC Form Rendering

To make the MVC form rendering work, before you start using it:

1. In the main layout, in the `<head>` section, add the following scripts.

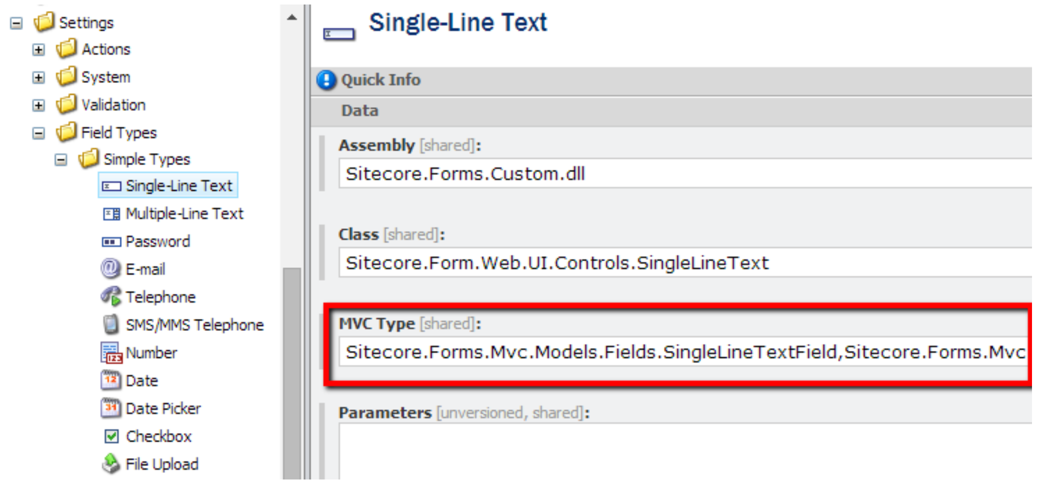
```
       <script src="~/sitecore/shell/Applications/Modules/Web Forms for Marketers/mvc/jquery-
1.8.2.min.js"></script>
       <script src="~/sitecore/shell/Applications/Modules/Web Forms for Marketers/mvc/jquery-ui-
1.8.24.js"></script>
       <script src="~/sitecore/shell/Applications/Modules/Web Forms for
Marketers/mvc/jquery.validate.js"></script>
       <script src="~/sitecore/shell/Applications/Modules/Web Forms for
Marketers/mvc/jquery.validate.unobtrusive.js"></script>
       <script src="~/sitecore/shell/Applications/Modules/Web Forms for
Marketers/mvc/Fields/sc.fieldsunobtrusive.js"></script>
       <script src="~/sitecore/shell/Applications/Modules/Web Forms for
Marketers/mvc/Fields/sc.fieldsevents-tracking.js"></script>
       <script src="~/sitecore/shell/Applications/Modules/Web Forms for
Marketers/mvc/Fields/sc.fieldsdate.js"></script>
       <script src="~/sitecore/shell/Applications/Modules/Web Forms for
Marketers/mvc/Fields/sc.fieldscaptcha.js"></script>
       <script src="~/sitecore/shell/Applications/Modules/Web Forms for
Marketers/mvc/Fields/sc.ajaxform.js"></script>
```

2. In the main layout, in the `<head>` section, add the following styles.

```
       <link href="~/sitecore/shell/Themes/Standard/Default/WFM/mvc/Fields/Default.css"
rel="stylesheet">
       <link href="~/sitecore/shell/Themes/Standard/Default/WFM/mvc/Fields/Colors/Default.css"
rel="stylesheet">
       <link href="~/sitecore/shell/Themes/Standard/Default/WFM/mvc/Fields/Custom.css"
rel="stylesheet">
       <link href="~/sitecore/shell/Themes/Standard/Default/WFM/mvc/base/jquery.ui.all.css"
rel="stylesheet">
```

3.15.3 How Fields Are Rendered and How to Change HTML

When you open the *Field Type* item, you can see a new *MVC Type* field that refers to a relevant MVC field model.



Assembly [shared]: Sitecore.Forms.Custom.dll

Class [shared]: Sitecore.Form.Web.UI.Controls.SingleLineText

MVC Type [shared]: Sitecore.Forms.Mvc.Models.Fields.SingleLineTextField,Sitecore.Forms.Mvc

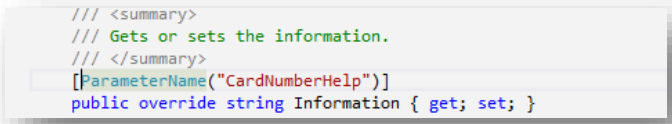
Parameters [unversioned, shared]:

Base class for each field is the `FieldModel`, `FieldModel`'s base class is `Sitecore.Forms.Core.Data.FieldItem`. So you get all field item information in a model and can extend it if you need. By default, the `FieldModel` view is used for all field types. If you need to have an advanced view, a user can define an editor template for this type of a field. You can find all project views in the `/Website/Views/Form` folder.

You can change each field view by editing a relevant field view *`.cshtml` file.

By default, field properties values are initialized from the **Parameters** and **Localized Parameters** fields if the property name coincides with the parameter name. For example, the `CssClass` model property will be initialized with the *CssClass* parameter value if it exists.

If you want a property to be initialized from a parameter with another name, use `ParameterNameAttribute`, for example:



```
/// <summary>
/// Gets or sets the information.
/// </summary>
[ParameterName("CardNumberHelp")]
public override string Information { get; set; }
```

3.15.4 How to Add a Custom Field Type

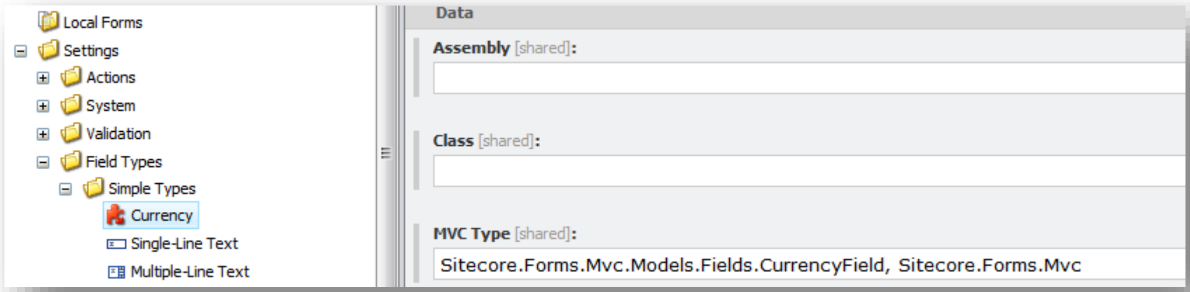
To create your custom field type, you can either extend the `FieldModel` class or customize the existing type. If `FieldModel.cshtml` view doesn't suit your needs, create an editor template for this field.

Custom Field Type

You can create a custom field type.

To create your custom field type:

1. Add a new field type item in the *Field Types* folder.

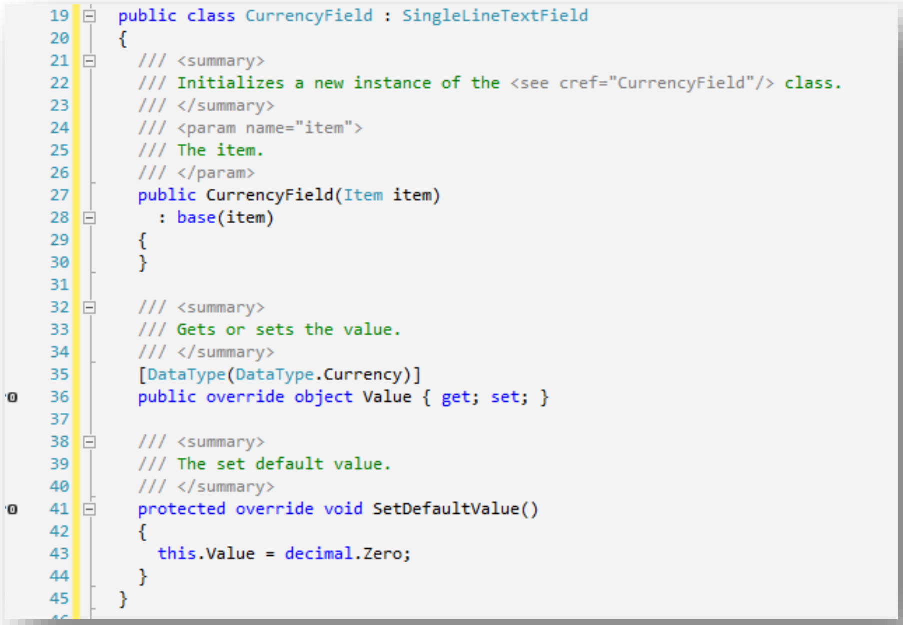


Assembly [shared]:

Class [shared]:

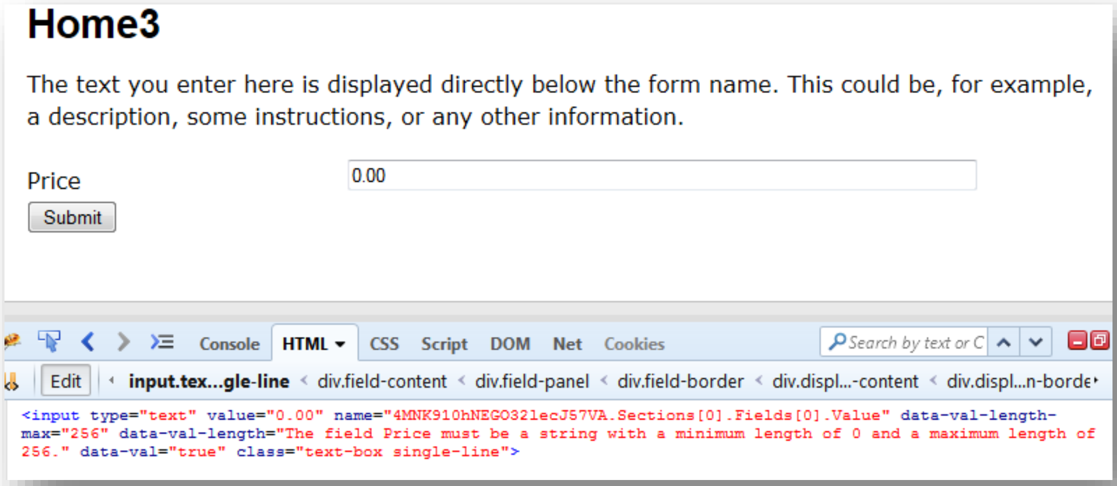
MVC Type [shared]: Sitecore.Forms.Mvc.Models.Fields.CurrencyField, Sitecore.Forms.Mvc

2. In the Visual Studio, in your project, add a new class `CurrencyField` that is located in the MVC Type namespace.



```
public class CurrencyField : SingleLineTextField
{
  /// <summary>
  /// Initializes a new instance of the <see cref="CurrencyField"/> class.
  /// </summary>
  /// <param name="item">
  /// The item.
  /// </param>
  public CurrencyField(Item item)
    : base(item)
  {
  }

  /// <summary>
  /// Gets or sets the value.
  /// </summary>
  [DataType(DataType.Currency)]
  public override object Value { get; set; }

  /// <summary>
  /// The set default value.
  /// </summary>
  protected override void SetDefaultValue()
  {
    this.Value = decimal.Zero;
  }
}
```

3. The `FieldModel.cshtml` view is used in this case for rendering of this type of a field.



Home3

The text you enter here is displayed directly below the form name. This could be, for example, a description, some instructions, or any other information.

Price 0.00

Submit

```
<input type="text" value="0.00" name="4MNK910hNEGO32lecJ57VA.Sections[0].Fields[0].Value" data-val-length-
max="256" data-val-length="The field Price must be a string with a minimum length of 0 and a maximum length of
256." data-val="true" class="text-box single-line">
```

Note
The `CurrencyField` class in this example is based on the `SingleLineTextField` class. (See the screenshot in step 2). The `Value` property in the `SingleLineTextField` class contains the `length` property validator. As a result, the `Value` property of the `CurrencyField` class has the same validator.

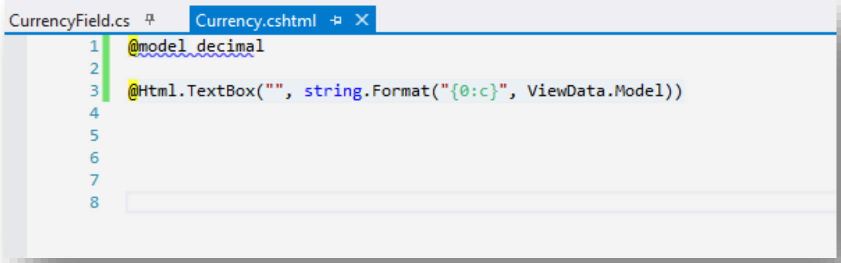
All validators from the basic classes are inherited in a child class.

Custom Field Value Rendering

You can also render `Value` in a different way by creating an Editor Template.

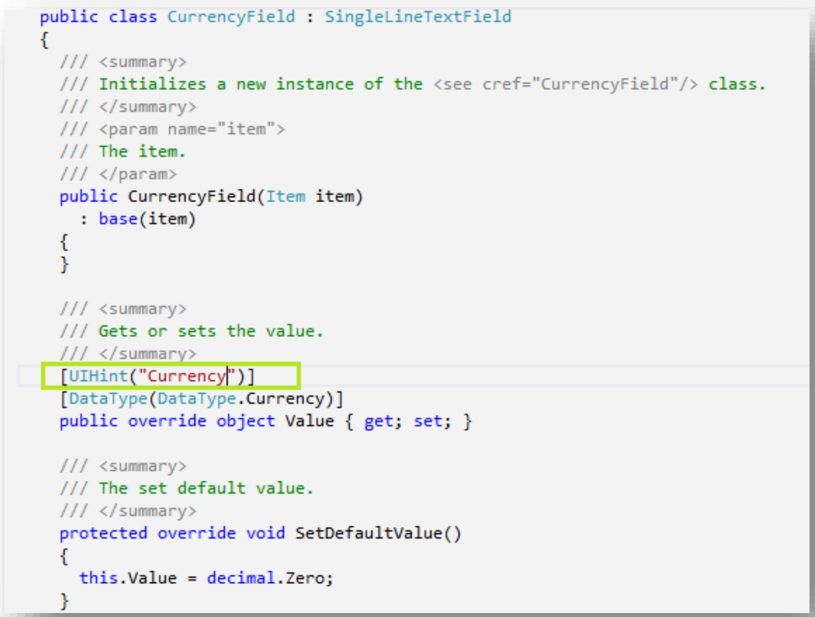
To create an example editor template:

1. In Visual Studio, create `Currency.cshtml`:



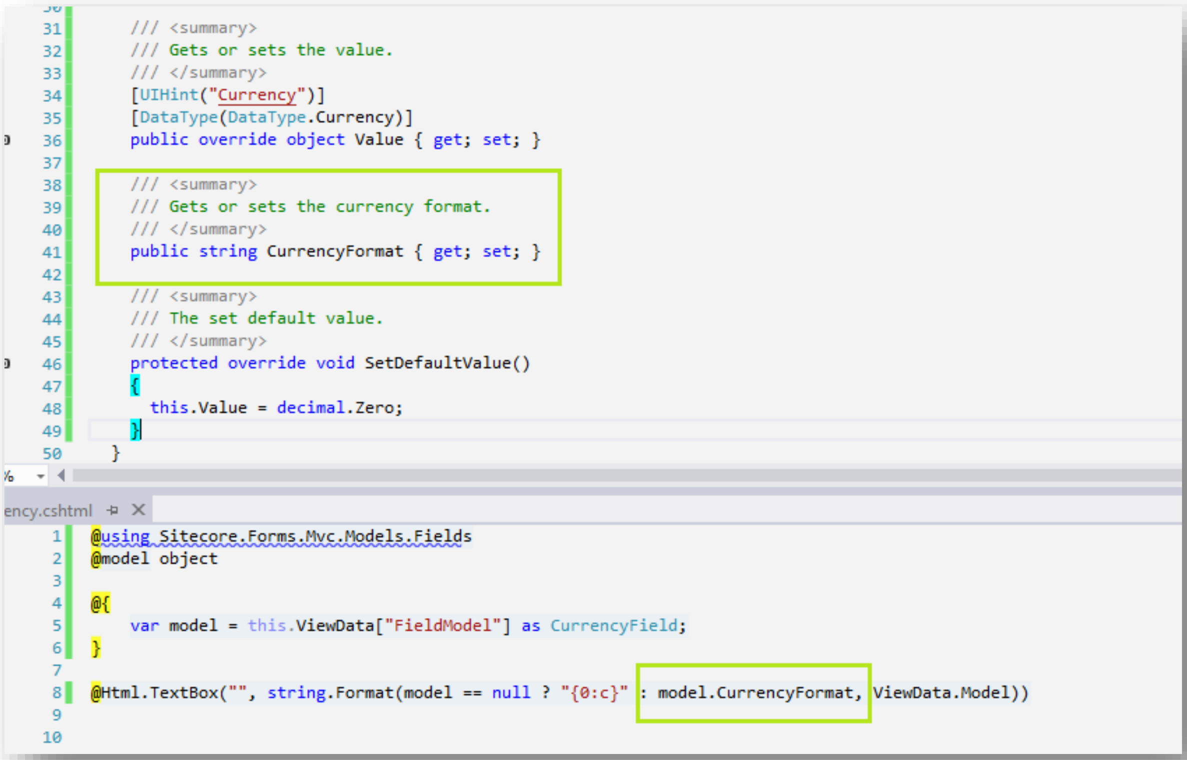
```
@model decimal

@Html.TextBox("", string.Format("{0:c}", ViewData.Model))
```

2. In the `CurrencyField` class, change your model by adding the `UIHint` attribute:



```
public class CurrencyField : SingleLineTextField
{
  /// <summary>
  /// Initializes a new instance of the <see cref="CurrencyField"/> class.
  /// </summary>
  /// <param name="item">
  /// The item.
  /// </param>
  public CurrencyField(Item item)
    : base(item)
  {
  }

  /// <summary>
  /// Gets or sets the value.
  /// </summary>
  [UIHint("Currency")]
  [DataType(DataType.Currency)]
  public override object Value { get; set; }

  /// <summary>
  /// The set default value.
  /// </summary>
  protected override void SetDefaultValue()
  {
    this.Value = decimal.Zero;
  }
```

3. If you need additional data to render the `Value` field from the field model object, use the `FieldModel` key to retrieve the field model from the `ViewData` object.



```
    /// <summary>
    /// Gets or sets the value.
    /// </summary>
    [UIHint("Currency")]
    [DataType(DataType.Currency)]
    public override object Value { get; set; }

    /// <summary>
    /// Gets or sets the currency format.
    /// </summary>
    public string CurrencyFormat { get; set; }

    /// <summary>
    /// The set default value.
    /// </summary>
    protected override void SetDefaultValue()
    {
      this.Value = decimal.Zero;
    }
  }
```

ency.cshtml

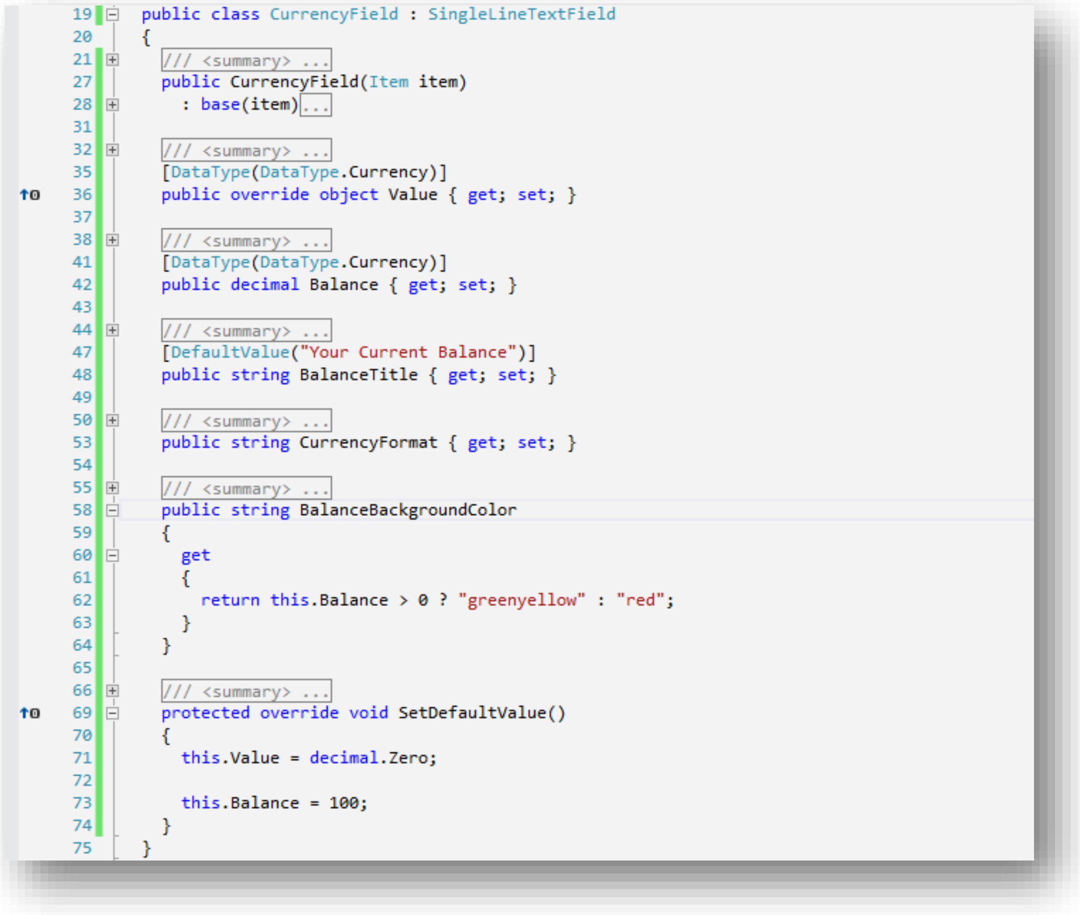
```
@using Sitecore.Forms.Mvc.Models.Fields
@model object

@{
    var model = this.ViewData["FieldModel"] as CurrencyField;
}

@Html.TextBox("", string.Format(model == null ? "{0:c}" : model.CurrencyFormat, ViewData.Model))
```

Custom Field View

If you want to create unique HTML for a field, you can create your custom field view.

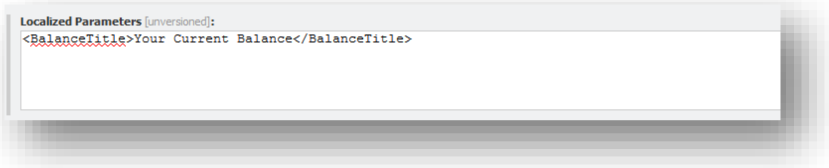
To create a custom field view:

1. Add the `Balance` and `BalanceTitle` properties to the `CurrencyField` class.



```
public class CurrencyField : SingleLineTextField
{
  /// <summary> ...
  public CurrencyField(Item item)
    : base(item)...

  /// <summary> ...
  [DataType(DataType.Currency)]
  public override object Value { get; set; }

  /// <summary> ...
  [DataType(DataType.Currency)]
  public decimal Balance { get; set; }

  /// <summary> ...
  [DefaultValue("Your Current Balance")]
  public string BalanceTitle { get; set; }

  /// <summary> ...
  public string CurrencyFormat { get; set; }

  /// <summary> ...
  public string BalanceBackgroundColor
  {
    get
    {
      return this.Balance > 0 ? "greenyellow" : "red";
    }
  }

  /// <summary> ...
  protected override void SetDefaultValue()
  {
    this.Value = decimal.Zero;

    this.Balance = 100;
  }
}
```

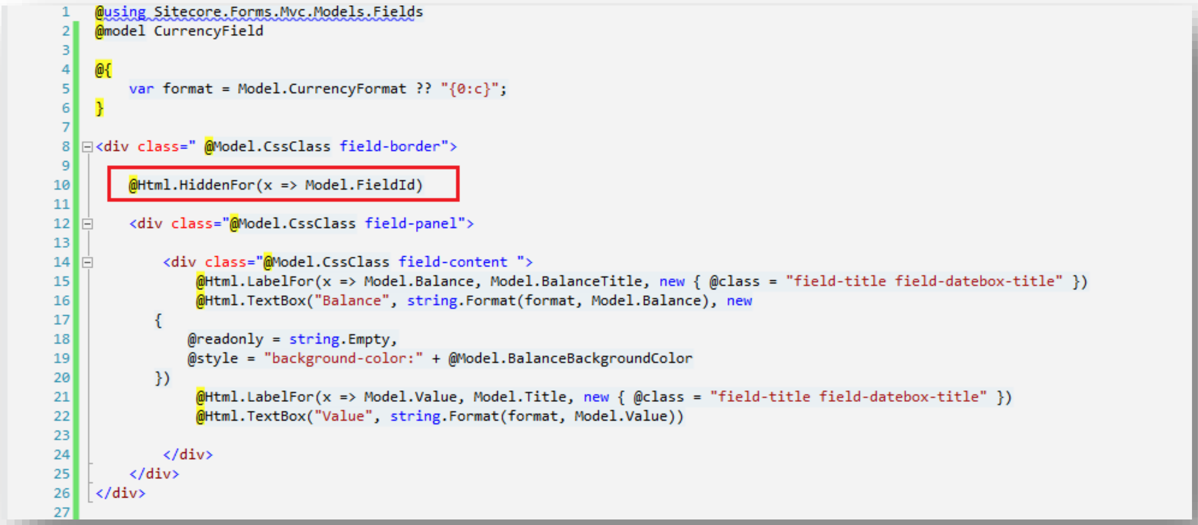
2. Insert the `BalanceTitle` value into the **Localized Parameters** field so that the `BalanceTitle` parameter can be localized.



Localized Parameters [unversioned]:

```
<BalanceTitle>Your Current Balance</BalanceTitle>
```

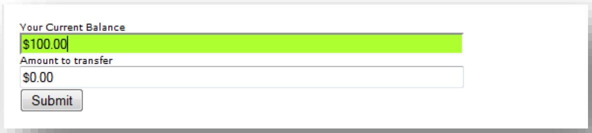
3. Your custom `CurrencyField` view will look as follows:



```
@using Sitecore.Forms.Mvc.Models.Fields
@model CurrencyField

@{
    var format = Model.CurrencyFormat ?? "{0:c}";
}

<div class=" @Model.CssClass field-border">

    @Html.HiddenFor(x => Model.FieldId)

    <div class="@Model.CssClass field-panel">

        <div class="@Model.CssClass field-content ">
            @Html.LabelFor(x => Model.Balance, Model.BalanceTitle, new { @class = "field-title field-datebox-title" })
            @Html.TextBox("Balance", string.Format(format, Model.Balance), new
        {
            @readonly = string.Empty,
            @style = "background-color:" + @Model.BalanceBackgroundColor
        })
            @Html.LabelFor(x => Model.Value, Model.Title, new { @class = "field-title field-datebox-title" })
            @Html.TextBox("Value", string.Format(format, Model.Value))

        </div>
    </div>
</div>

```

Important
While writing your custom view for a field model, always make the hidden input for the `FieldId` property rendered. This is required for client analytics events tracking logic.

This action results in the following:

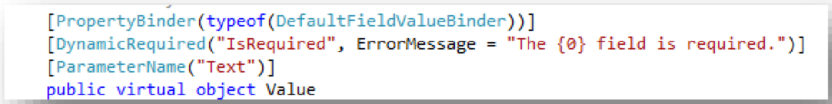


Your Current Balance
$100.00
Amount to transfer
$0.00
Submit

3.15.5 Field Validation

The MVC implementation of the Web Forms for Marketers module does not use validation items structure `/sitecore/system/Modules/Web Forms for Marketers/Settings/Validation` for field validation. Each field type contains predefined validations built-in in the model depending on the field type, for example: e-mail, phone number, or credit card validations.

By default, the `FieldModel Value` property contains one validation `DynamicRequiredAttribute` attribute.



```
[PropertyBinder(typeof(DefaultFieldValueBinder))]
[DynamicRequired("IsRequired", ErrorMessage = "The {0} field is required.")]
[ParameterName("Text")]
public virtual object Value
```

The type-specific validation attribute is set on a given model:

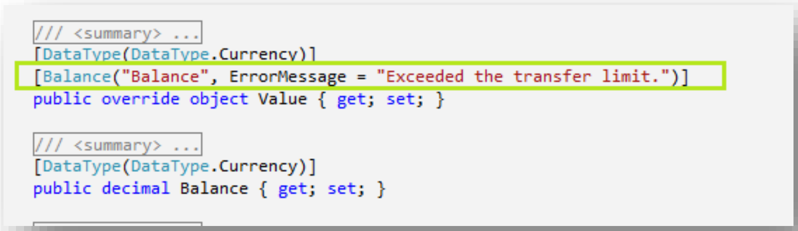
For more information about how to add a predefined validator to a regular expression, see the section *Validations*.

How to Add Additional Validations

To add an additional validation, you must create a validation attribute for your field.

For example if you want to create a validation that must validate the `CurrencyField Value` depending on `Balance`, you must perform the following actions:

1. Add the `Balance` validation attribute.



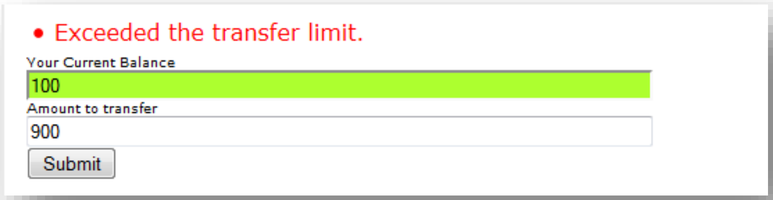
```
/// <summary> ...
[DataType(DataType.Currency)]
[Balance("Balance", ErrorMessage = "Exceeded the transfer limit.")]
public override object Value { get; set; }

/// <summary> ...
[DataType(DataType.Currency)]
public decimal Balance { get; set; }
```

2. The `BalanceAttribute` class can look as follows.

```
          /// <summary>
          /// The balance validator.
          /// </summary>
          public class BalanceAttribute : DynamicValidationBase
          {
            /// <summary>
            /// Initializes a new instance of the <see cref="BalanceAttribute"/> class.
            /// </summary>
            /// <param name="balanceProperty">
            /// The balance property.
            /// </param>
            public BalanceAttribute(string balanceProperty)
            {
              Assert.ArgumentNotNullOrEmpty(balanceProperty, "balanceProperty");
                this.BalanceProperty = balanceProperty;
            }

            /// <summary>
            /// Gets or sets the balance property.
            /// </summary>
            public string BalanceProperty { get; set; }

            /// <summary>
            /// The get client validation rules.
            /// </summary>
            /// <param name="metadata">
            /// The metadata.
            /// </param>
            /// <param name="context">
            /// The context.
            /// </param>
            /// <returns>
            /// The <see cref="IEnumerable"/>.
            /// </returns>
            public override IEnumerable<ModelClientValidationRule>
GetClientValidationRules(ModelMetadata metadata, ControllerContext context)
            {
              yield break;
            }

            /// <summary>
            /// The is valid.
            /// </summary>
            /// <param name="value">
            /// The value.
            /// </param>
            /// <param name="validationContext">
            /// The validation context.
            /// </param>
            /// <returns>
            /// The <see cref="ValidationResult"/>.
            /// </returns>
            protected override ValidationResult IsValid(object value, ValidationContext 
validationContext)
            {
              var fieldModel = this.GetModel(validationContext);

              var balance = fieldModel.GetPropertyValue<decimal>(this.BalanceProperty);
              var valueString = value as string;

              if (string.IsNullOrEmpty(valueString))
              {
                   return ValidationResult.Success;
              }

              decimal valueDecimal;

              decimal.TryParse(valueString, out valueDecimal);

              if (decimal.TryParse(valueString, out valueDecimal) && decimal.Parse(valueString) 
<= balance)
              {
                return ValidationResult.Success;
              }

              return new ValidationResult(string.Format(CultureInfo.CurrentCulture, 
this.GetErrorMessageTemplate(fieldModel), fieldModel.Title));
            }
```

Note
To create your custom validation attribute, you can override either the `Sitecore.Forms.Mvc.Validators.DynamicValidationBase` class or the `System.ComponentModel.DataAnnotations.ValidationAttribute` class.

3. When implemented, you custom validation can look in the following way:


- Exceeded the transfer limit.

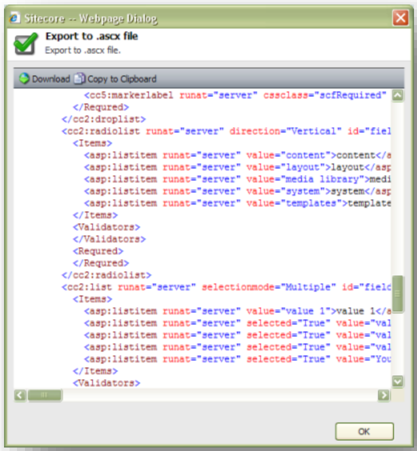
Your Current Balance
100
Amount to transfer
900

Note
This validation is a server one. To make the validation work on your client, override the `GetClientValidationRules` method.

3.16 How to Export to ASCX

Converting a web form to an .ascx file is a good solution when you want to change the behavior or the appearance of a particular web form without affecting all other web forms.

In the **Form Designer**, click **Export to ascx** to convert the form to an .ascx file. The code is displayed in the **Export to ascx file** dialog box.



```
        <cc5:markerlabel runat="server" cssclass="scfRequired"
      </Requred>
    </cc2:droplist>
    <cc2:radiolist runat="server" direction="Vertical" id="fiel
      <Items>
        <asp:listitem runat="server" value="content">content</a
        <asp:listitem runat="server" value="layout">layout</asp
        <asp:listitem runat="server" value="media library">medi
        <asp:listitem runat="server" value="system">system</asp
        <asp:listitem runat="server" value="templates">template
      </Items>
      <Validators>
      </Validators>
      <Requred>
      </Requred>
    </cc2:radiolist>
    <cc2:list runat="server" selectionmode="Multiple" id="field
      <Items>
        <asp:listitem runat="server" value="value 1">value 1</a
        <asp:listitem runat="server" selected="True" value="val
        <asp:listitem runat="server" selected="True" value="val
        <asp:listitem runat="server" selected="True" value="val
        <asp:listitem runat="server" selected="True" value="You
      </Items>
      <Validators>
```

Click **Download** to store the code as an `.ascx` file.

Note
After you have converted a web form to the `.ascx` file, do not delete the web form item in the Content Tree. This item is required for the analytics reports.

3.16.1 How to Show or Hide a Field Depending on another Field Value

In this example, we will show how you can use exporting to an `.ascx` file. For example, you want a web form to display or hide a field depending on another field value.

To implement this, perform the following actions:

1. Create a web form that contains a *Single-Line Text* field and a *Check box* field.
2. Export the web form to `.ascx` file.
3. Add the exported web form to a web page.

   For more information about this, see the section *How to Add an ASCX Control to the Page* section.
4. In the web control file, find the <input id> tags of the text and checkbox fields, for example:

   Text field ID `<input id="WebUserControl1_field_5339820707D14FAC88D66DCC8F81EB01"`

   Checkbox field ID `<input id="WebUserControl1_field_F2ACADD39B3C46A4A3036C63C0D60C3C"`
5. Create a client script that shows or hides the text field depending on the selected checkbox value:

```
        <script type="text/javascript">
          var checkbox =
document.getElementById("WebUserControl1_field_F2ACADD39B3C46A4A3036C63C0D60C3C");
          var textbox =
document.getElementById("WebUserControl1_field_5339820707D14FAC88D66DCC8F81EB01").parentNode.pare
ntNode;
          textbox.style.display = "none";
          checkbox.onclick = function () {
            if (checkbox.checked) {
              textbox.style.display = "block";
            }
            else {
              textbox.style.display = "none";
            }
          }
        </script>
```

6. Place this code at the end of the `.ascx` file.

3.16.2 How to Add an ASCX Control to the Page

After you converted a web form to an ASCX file, you can add an ASCX control to the page.

To add an ASCX control to the page:

1. In the layouts folder, create a new `default.aspx` page.

```
        <%@ Page Language="C#" AutoEventWireup="true" %>
        <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

        <html xmlns="http://www.w3.org/1999/xhtml" >
        <head runat="server">
            <title>Untitled Page</title>
        </head>
        <body>
            <form id="form1" runat="server">
            </form>
        </body>
        </html>
```

2. Click **Export to ascx** to export the form you need to the layouts\ folder. Use the `forms.ascx` format for the name of the file.

3. Add the following changes to the `default.aspx` page:

```
        <%@ Page Language="C#" AutoEventWireup="true"  %>
        <%@ Register Src="form.ascx" TagName="SimpleForm" TagPrefix="uc1" %>
        <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

        <html xmlns="http://www.w3.org/1999/xhtml" >
        <head runat="server">
            <title>Untitled Page</title>
        </head>
        <body>
            <form id="form1" runat="server">
            <div>
                <uc1:SimpleForm ID="WebUserControl1" runat="server" />
            </div>
            </form>
        </body>
        </html>
```

In the browser, type http://localhost/layouts/default.aspx.

After you make any changes to the `.ascx` file, the module does not verify that the form works correctly.

3.17 How to Re-Install the Module

Sometimes you have to re-install the Web Forms for Marketers module to make sure all the files and items are correct. The files and items could be corrupted during an unsuccessful upgrade.

Note
When you re-install the module, all the save action parameters reset to default values.

To re-install the module and save all the created web forms:

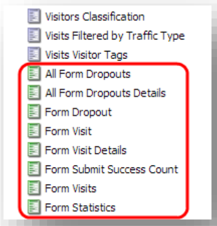
1. Create a Sitecore package that contains all the items of the created web forms.
2. Back up the module database.
3. If you modified the `Sitecore.Forms.config` file, back up this file.
4. Install the Web Forms for Marketers installation package.
   - Choose **Overwrite all** when prompted.
   - Choose **Continue always** when prompted.
5. When the installation is finished, install a package with web form items.
6. Restore the module database.
7. Restore the `Sitecore.Forms.config` file.
8. Configure the save action parameters because those parameters were reset to default values. For example, in the *Send Email Message* save action, set the host parameter.

3.18 How to Uninstall the Module

You can uninstall the module by removing all the related files and items.

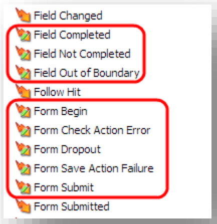
To uninstall the Web Forms for Marketers module, perform the following actions:

1. In the Master database, delete the following items:
   - /sitecore/layout/Renderings/Modules/Web Forms for Marketers
   - /sitecore/media library/Web Forms for Marketers
   - /sitecore/system/Modules/Web Forms for Marketers
   - /sitecore/system/Marketing Center/Goals/Leave a message
   - /sitecore/system/Marketing Center/Goals/Tell a Friend
   - /sitecore/system/Settings/Rules/Web Forms for Marketers Conditions
   - /sitecore/system/Settings/Analytics/Reports/Subreports/Web Forms for Marketers Reports
   - /sitecore/system/Settings/Analytics/Reports/Subreports/Web Forms for Marketers Detailed Reports
   - /sitecore/templates/Branches/Web Forms for Marketers
   - /sitecore/templates/Web Forms for Marketers
   - All report items related to web forms in the */sitecore/system/Settings/Analytics/Reports SQL Queries* folder with the icon



Visitors Classification
Visits Filtered by Traffic Type
Visits Visitor Tags
All Form Dropouts
All Form Dropouts Details
Form Dropout
Form Visit
Form Visit Details
Form Submit Success Count
Form Visits
Form Statistics

   - All page event items related to web forms in the */sitecore/system/Settings/Analytics/Page Events* folder with the icon



Field Changed
Field Completed
Field Not Completed
Field Out of Boundary
Follow Hit
Form Begin
Form Check Action Error
Form Dropout
Form Save Action Failure
Form Submit
Form Submitted

   - All goal items related to web forms in the */sitecore/system/Marketing Center/Goals* folder

2. In the *Core* database, delete the following items:
    - /sitecore/content/Applications/Content Editor/Ribbons/Chunks/Forms
    - /sitecore/content/Applications/Content Editor/Ribbons/Contextual Ribbons/Forms
    - /sitecore/content/Applications/Content Editor/Ribbons/Strips/Presentation/Forms
    - /sitecore/content/Applications/Modules/Web Forms for Marketers
    - /sitecore/content/Applications/WebEdit/Custom Experience Buttons/Edit Form
    - /sitecore/content/Documents and settings/All users/Start menu/Programs/Web Forms for Marketers
3. In the file system, delete the following files:
    - \App_Config\Include\Captcha.config
    - \App_Config\Include\Sitecore.Forms.config
    - \bin\ MSCaptcha.dll
    - \bin\ Sitecore.Forms.Core.dll
    - \bin\ Sitecore.Forms.Custom.dll
    - \bin\ System.Data.SQLite.dll
    - \bin_x64\ System.Data.SQLite.dll
    - WFFM databases in the \Data folder
    - \layouts\system\ VisitorIdentificationExtension.aspx
    - \sitecore\shell\Applications\Modules\Web Forms for Marketers
    - \sitecore\shell\Themes\Standard\Default\WFM
    - \sitecore\shell\Themes\Standard\Default\FormBuilder.css
    - \sitecore\shell\Themes\Standard\Default\FormDataViewer.css
    - \sitecore\shell\Themes\Standard\Default\MultiTreeView.css
    - \sitecore\shell\Themes\Standard\Default\Placeholder.css
    - \sitecore\shell\Themes\Standard\Firefox\WFM
    - \sitecore\shell\Themes\Standard\Firefox\ FormBuilder.css
    - \sitecore modules\Shell\Web Forms for Marketers
    - \sitecore modules\Web\Web Forms for Marketers
4. Publish the website.