Sitecore CMS 7.1

Rules Engine Cookbook

Rules Engine and Conditional Rendering Tips and Techniques for Developers

Table of Contents

Chapter 1

Introduction

Sitecore administrators and developers should read this document before using the Sitecore rules engine. You can use the rules engine to manipulate insert options dynamically, to handle system events, to validate items, and for conditional rendering (sometimes referred to as personalization, behavioral targeting, multivariate testing, A/B testing, or behavioral marketing).

For more information about multivariate testing, see the *Marketing Operations Cookbook*.

This document contains the following chapters:

- **Chapter 1 — Introduction**
  This introduction to the manual.
  This chapter describes the concepts behind the Sitecore rules engine, including an overview of the rules engine, scripts, conditions, actions, and rules.
- **Chapter 2 — Conditional Rendering**
  This chapter explains how to implement a conditional rendering that allows runtime manipulation of presentation components.
- **Chapter 3 — Implementing a Conditional Rendering Rule**
  This chapter describes how to implements a conditional rendering rule and contains some code samples.

1.1 Rules Engine Overview

The Sitecore rules engine applies the logic that is defined in actions when the specific conditions defined in a rule evaluate to *True*. The rules engine provides user interfaces to control Sitecore features including insert options, conditional rendering, and personalization. These rules are used extensively in engagement plans.

For more information about the layout engine, see the Presentation Component Reference manual.

For more information about insert options, see the Data Definition Reference manual.

All of the conditions, actions, and rules are stored in the `/sitecore/system/Settings/Rules` folder.

The Sitecore Rule Set Editor allows you to combine these different elements — conditions and actions — to make rules that you can use in personalization, engagement plans, and so on.

1.1.1 Conditions

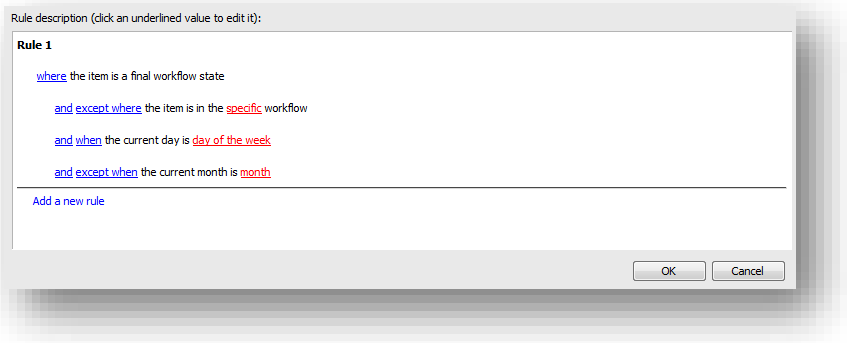
Each condition contains logic that determines whether or not that condition is true. For example, the *where the item is locked by me* condition is true when I have locked the item.

A condition consists of a condition definition item and the .NET class that implements that condition.

Condition Text

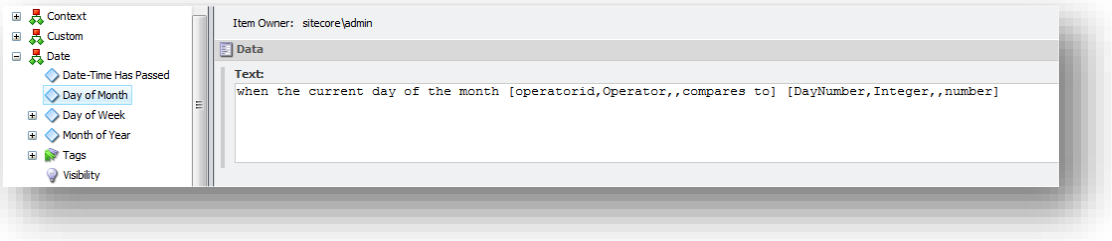
When you select a condition in the user interface, the text that you see is contained in the condition definition item, in the **Data** section, the **Text** field. Some conditions contain tokens that are replaced by parameters that you can change when you select the condition as part of a rule.

When you select a condition, Sitecore inserts links in the words `when`, and `where`. If you click the link, Sitecore reverses the condition, alternating between `when` and `except when`, or `where` and `except where`.



Note
If you reverse the condition, the rules engine automatically reverses the result of the condition logic. The code that implements the individual conditions does not take your selection into account.

In the condition text, the tokens in square brackets `[]` enable links that allow you to specify the parameters for the condition.



The brackets contain four comma separated positional parameters:

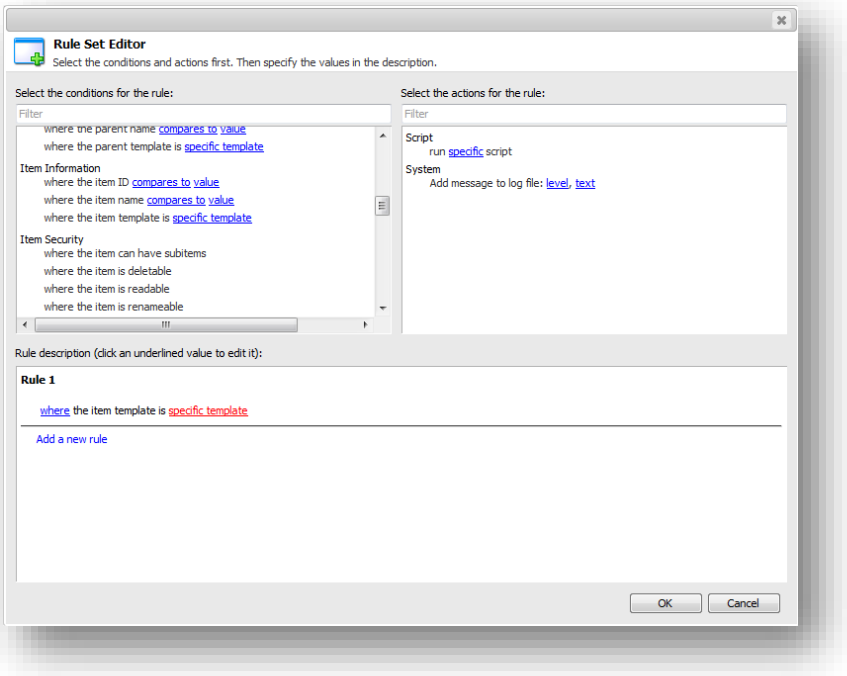
1. The name of the property that the parameter sets in the .NET class that implements the condition.
2. The name of an item in the `/Sitecore/System/Settings/Rules/Definitions/Macros` folder that contains the predefined functionality, or an empty string. For example, to activate a user interface that allows the user to select an integer, specify the *Integer* macro.
3. URL parameters to the user interface activated by the macro specified by the previous parameter, or an empty string. For example, to specify the `/Sitecore/Content/Home` item as the root for a selection tree, enter `root=/sitecore/content/home`.
4. The text that is displayed if the user does not specify a value for the parameter.

Tip
Investigate the default condition definition items for more information about the syntax of the **Text** field.

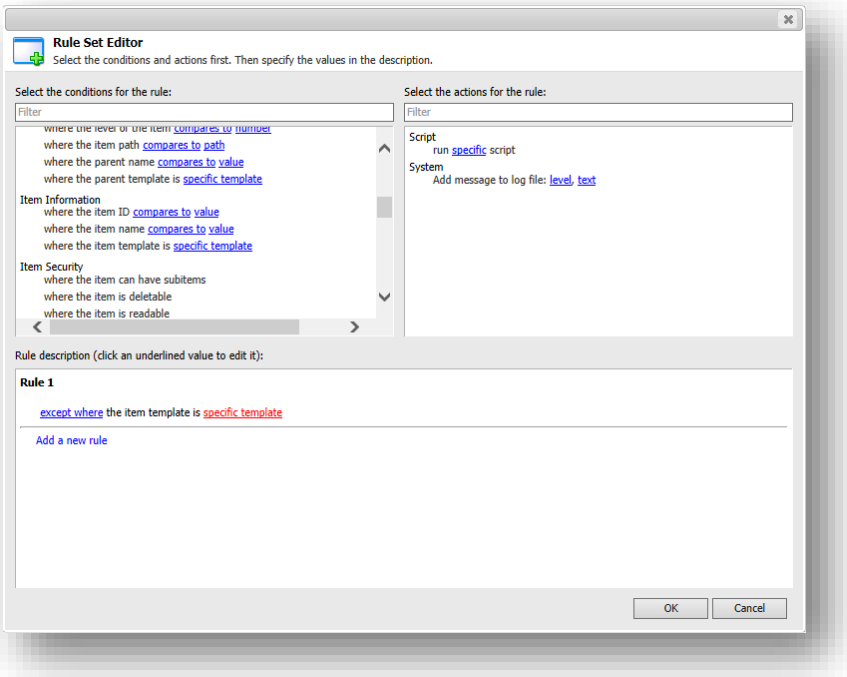
For example, the *where the item template is specific template* condition evaluates to *True* when an item is associated with a specific data template. The text for this condition is:

```
where the item template is [templateid,Tree,root=/sitecore/templates,specific template]
```

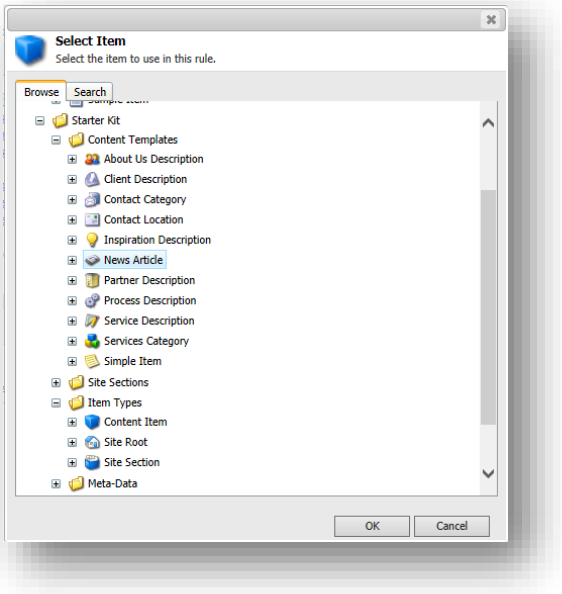
In the Rule Set Editor where you select the conditions and edit the parameters, Sitecore translates the word *where* and the tokens in square brackets and renders the text as shown in the following image:



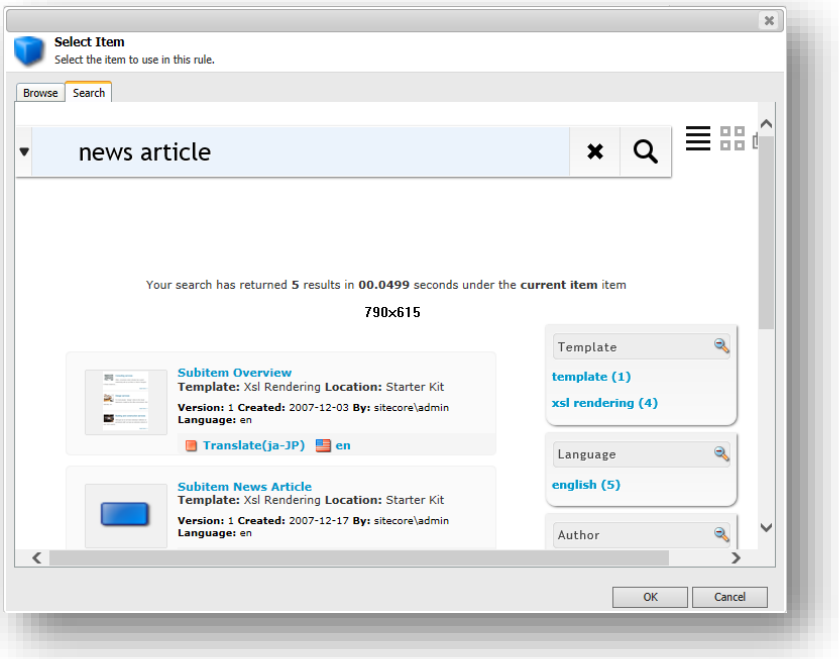
In the **Rule description** field, you can select a template and reverse the condition. If you click the word *where*, Sitecore reverses the condition and changes it to *except where*.



If you click the words *specific template*, Sitecore prompts you to select a data template.



You can also use the **Search** tab to locate the item you want.



The selection that you make in this dialog box controls the `TemplateID` property of the class that implements the condition.

For more information about defining condition parameters, see the section *Rules*.

How to Implement a Condition

To implement a condition:

1. In the Visual Studio project, create the condition class by inheriting from:

Condition	Function
`Sitecore.Rules.Conditions.WhenCondition`,	For Boolean conditions
`Sitecore.Rules.Conditions.StringOperatorCondition`	For comparing string values
`Sitecore.Rules.Conditions.OperatorCondition`,	For comparing other types of values.

Tip
You can save time by duplicating an existing condition with similar functionality.

2. In the condition class, implement the `Execute()` method to indicate whether the condition is `true` or `false`.
3. In the **Content Editor**, select the item under which you want to store the condition definition item.

   The conditions must be stored under the appropriate descendant of the `sitecore/system/Settings/Rules/Definitions/Elements` item.
4. Insert a condition definition item based on the

   `/sitecore/templates/System/Rules/Condition` data template. Name the condition

   definition item after the .NET class that implements the condition.
5. In the condition definition item, in the **Data** section, in the **Text** field, enter some text as described in the section *Condition Text*.
6. In the condition definition item, in the **Script** section, in the **Type** field, enter the type signature of the .NET condition class.

You can now use this conditional rendering in the rules that you create. For more information about configuring conditions in rules, see the section *Conditional Rendering Rules*.

1.1.2 Actions

Actions contain the logic that should be implemented when one or more of the conditions in a rule are true. For example, the *Add Insert Option* action adds an item to the insert options for an item.

An action consists of an action definition item and the .NET class that implements that action.

Note
When there is no user interface, actions can sometimes function as scripts. For more information, see the section *Scripts*. Scripts can also function as actions.

Action Text

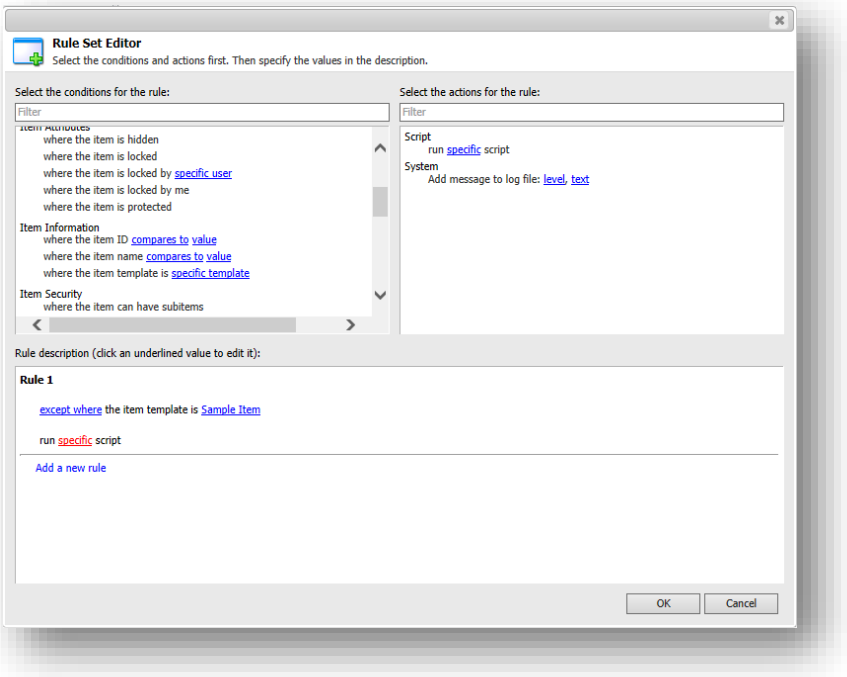
In user interfaces that allow the user to select actions and enter action parameters, Sitecore uses the value in the **Text** field in the **Data** section of the action definition items. For more information about the format of the **Text** field, see the section *Condition Text*.

Tip
For more information about the syntax of the **Text** field, investigate the default action definition items.

For example, the *Run Script* action invokes this script:

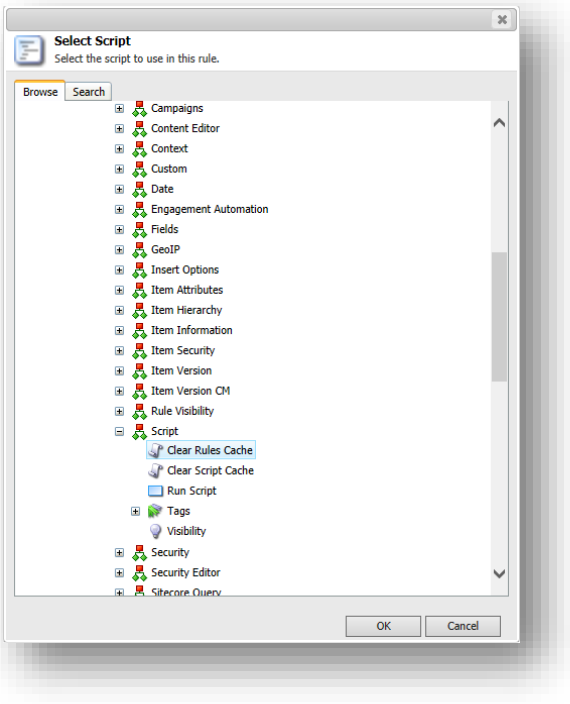
```
run [scriptid,Script,,specific] script
```

In the user interface that allows the user to enter action parameters, Sitecore translates the tokens in square brackets and renders the text as shown in the **Rule description** field.



For more information about defining rule parameters, see the section *Rules*.

Click the *specific* link and the **Select Script** dialog box opens where you can select a script.



When you select a script, this dialog controls the `ScriptID` property of the class that implements the action.

Note
You can configure a script as an action.

For more information about scripts, see the section *Scripts*.

How to Implement an Action

To implement an action:

1. In the Visual Studio project, create a class that inherits from the `Sitecore.Rules.Actions.RuleAction<T>` class, and implement the `Apply()` method.
2. In the **Content Editor**, navigate to the `/sitecore/system/Settings/Rules/Definitions/Elements` folder and select the appropriate element under which you want to store the action definition item.

   Note
   The actions must be stored under the appropriate descendant of the `sitecore/system/Settings/Rules/Definitions/Elements` item.

3. Use the `System/Rules/Action` data template to insert an action definition item. Name the action definition item after the .NET class that implements the action.
4. In the action definition item, in the **Data** section, in the **Text** field, enter the text for the action.

   For more information about action texts, see the section *Action Text*.

5. In the action definition item, in the **Script** section, in the **Type** field, enter the signature of the .NET class that implements the action.
6. Configure one or more rule definition items to reference the action definition item as described in the section *Conditional Rendering Rules*.

You can now use this action in different types of rules.

1.1.3 Scripts

Scripts contain logic that implements specific functions. A script consists of a script definition item that contains inline code or references the .NET class that implements that script.

The **Script** section of a script definition item contains the following fields:

- **Type** — The.NET class signature (`Namespace.Class, assembly`).
- **Code** — Inline code in any of the languages supported by the .NET framework.
- **References** — List of .NET assemblies to reference, separated by commas (“,”).
- **Language** — A .NET language specifier, such as `CSharp` for C#.

Important
Enter a value in the **Type** field, or a value in the **Code**, **References**, and **Language** fields. Do not enter values in all four fields.

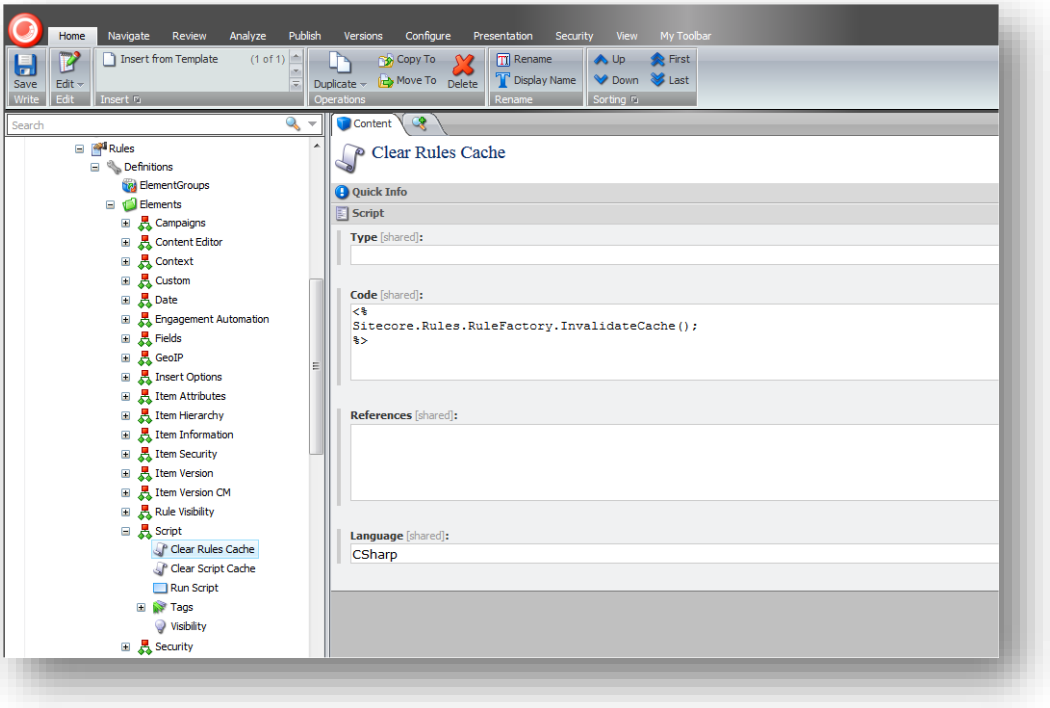
In most cases, you can use an action in place of a script.

When these fields appear in other data templates, they serve the purpose described in this section.

If the value of the **Code** field begins with `<%` and ends it with `%>`, Sitecore wraps the code like this:

```
namespace Sitecore
{
  public class DefaultClass
  {
    public void DefaultMethod()
    {
      // Contents of <% %>
    }
  }
}
```

For example, the *Clear Rules Cache* script clears all the caches that are related to the rules engine.



For more information about actions, see the section *Actions*.

How to Implement a Script

To implement a script:

1. In the **Content Editor**, navigate to the `/sitecore/system/Settings/Rules/Definitions/Elements` folder and select the item under which you want to store the script definition item.

   The script definition item must be stored in this folder.

2. Use the `System/Rules/Script` data template to insert a script definition item.
3. Give the script definition item a name that clearly indicates what the script does.
4. In the script definition item, in the **Data** section, in the **Type** field, enter the .NET class signature, or enter values in the **Code**, **References**, and **Language** fields.

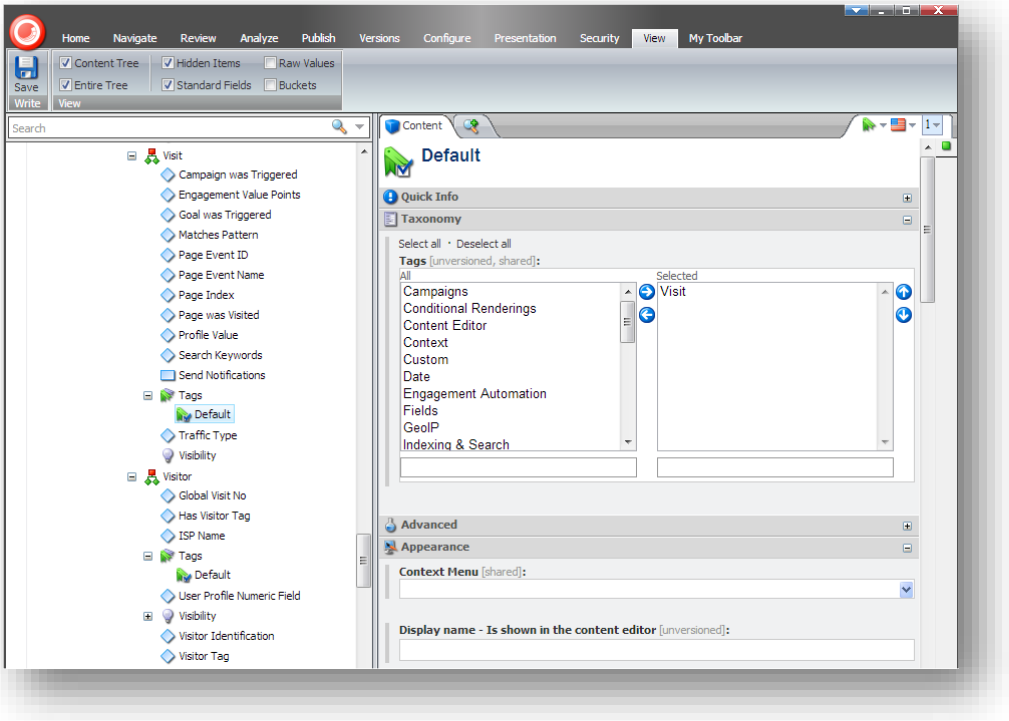
1.1.4 Specifying where Rule Elements should be Displayed

When you create a condition or an action, you must determine where it should be displayed. The Rule Set Editor should only display each element in contexts where it makes sense to use the element in a rule.

If the element relates to conditional rendering, it should only be displayed in the appropriate places. Similarly if the condition or action relates to visits it should only appear in contexts where it makes sense to use this rule element to create a rule about visits to your website.

To help you organize the rule elements, Sitecore groups the rule elements into element categories in the content tree For example, all the actions and conditions that relate to visitors are grouped together in the

content tree at `/sitecore/system/Settings/Rules/Definitions/Elements/Visit`.



In the previous example, the *Visit* tag is the default tag for this element category and this means that these rule elements are displayed in the Rule Set Editor when you are designing a rule that evaluates a visit.

After you create elements — actions and conditions — that can be used to create a rule, you must specify where they should be displayed.

To specify where they should be displayed, you can:

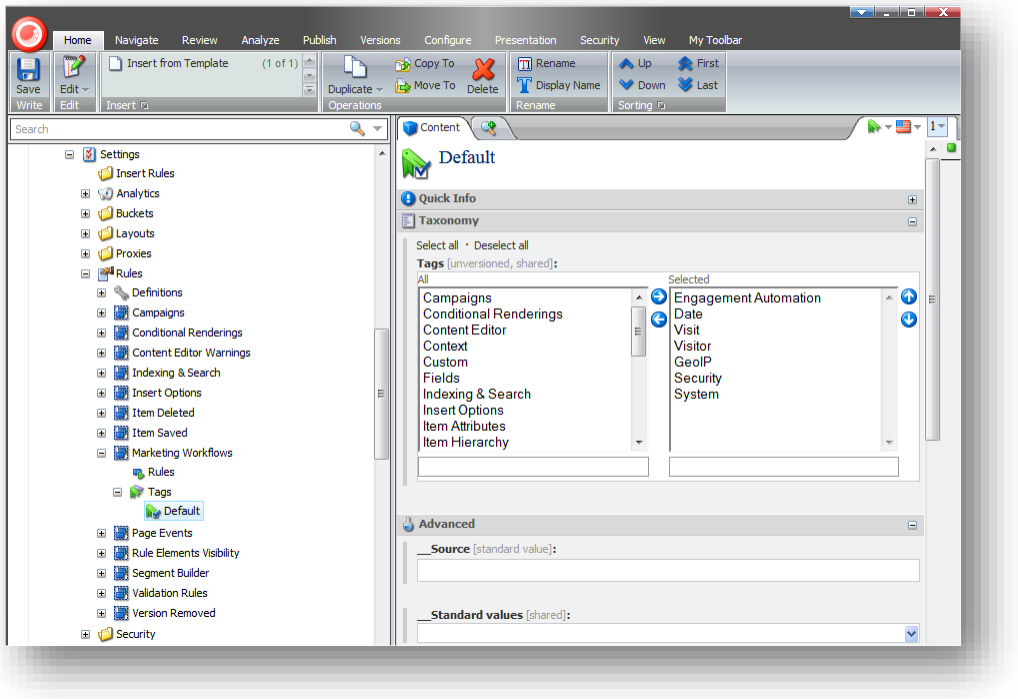
- Use an existing tag.
- Create a new tag.

To use an existing tag, store the rule elements under the appropriate element folder.

To create a new tag and assign it to a rule:

1. Navigate to the `/sitecore/system/Settings/Rules/Definitions/Tags` item.
2. On the **Home** tab, in the **Insert** group, click **Tag** and give the new tag a suitable name.
3. In the `/sitecore/system/Settings/Rules/Definitions/Elements` folder, add this tag to the rules element category that you created for the new rule or add it to an existing rules category.

4. In the `/sitecore/system/Settings/Rules` folder, select the appropriate rules context folder.



5. In the `Tags/Default` item for this rule context item, add the new tag to the **Selected** field.

The new tag is now linked to the rule.

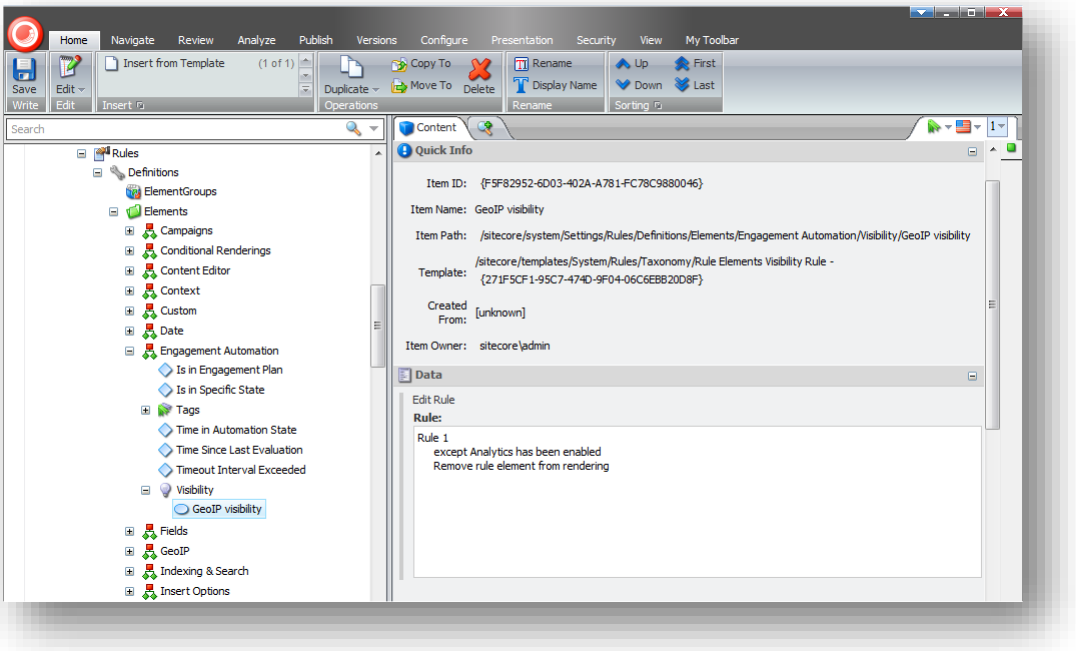
You can also implement a visibility rule that specifies that this rule should only be displayed when some extra conditions are met.

You can assign a visibility rule to rule tags and to rule elements categories.

To implement a visibility rule for a rule element category:

1. Navigate to the `/sitecore/system/Settings/Rules/Definitions/` item.
2. Expand the appropriate rule element category and select the Visibility item.
3. On the **Home** tab, in the **Insert** group, click Rules Element Visibility Rule and give the rule a suitable name.
4. Select the Rules Element Visibility Rule item and in the Data section in the Rule field click Edit Rule.
5. In the Rule Set Editor, create the visibility rule that you want to implement.

For example, we have already implemented a visibility rule that specifies that certain rules should only be visible when Analytics is enabled in your Sitecore installation.



1.2 Rules

Rules can associate one or more actions with one or more conditions, including parameter values for both actions and conditions. For example, a conditional renderings rule processes multivariate tests (action) if multivariate tests are enabled (condition). While it would be illogical in this case, a parameter could reverse the condition — process multivariate tests unless the multivariate tests are enabled.

You can use logical operators such as `and` and `or` to only invoke actions under specific combinations of conditions.

Types of Rules

In addition to insert options, insert rules, and the `uiGetMasters` pipeline, you can implement insert rules to control the effective insert options at runtime. You configure insert options rules beneath the `/Sitecore/System/Settings/Rules/Insert Options/Rules` item.

For more information about insert options, see the *Data Definition Reference* manual.

Note
Sitecore uses event handlers to invoke rules. To determine when rules are executed for events, investigate the corresponding event handlers.

- Use insert rules when you need to select insert options logic for specific items.
- Use the `uiGetMasters` pipeline for insert options logic that applies to all items.
- Use insert options rules to allow an administrator to configure insert options using variable logic.

In addition to the `item:deleting` and `item:deleted` events and the `uiDeleteItems` pipeline, you can implement item deleted rules to configure actions that are invoked after a user deletes items that match specific criteria. Configure item deleted rules beneath the `/Sitecore/System/Settings/Rules/Item Deleted/Rules` item.

In addition to the `item:saving` and `item:saved` events and the `saveUI` pipeline, you can implement item saved rules to configure actions that are invoked after a user saves an item that matches specific criteria. Configure item deleted rules beneath the `/Sitecore/System/Settings/Rules/Item Saved/Rules` item.

In addition to the `item:versionRemoving` and `item:versionRemoved` events and the `deleteVersionsUI` pipeline, you can implement version removed rules to configure actions that are invoked after a user deletes a version of an item that matches specific criteria. Configure version removed rules beneath the `/Sitecore/System/Settings/Rules/Version Removed/Rules` item.

In additional to the validation rules, which are stored under the `/sitecore/system/Settings/Validation Rules` item, you can implement a validation item rule which uses the Sitecore Rules Engine to validate the item. Configure the validation item rules beneath the `/sitecore/system/Settings/Rules/Validation Rules/Rules` item.

Sitecore uses different data templates for conditional rendering, but they follow the same general rules.

For more information about conditional rendering, see the section *Conditional Rendering*.

How to Implement a Rule

To implement a rule:

1. In the **Content Editor**, select the item under which you want to store the rule definition item.

   Store the rule definitions that handle deletion under the `/Sitecore/System/Settings/Rules/Item Deleted/Rules` item.

   Store the rule definitions that handle saving under the `/Sitecore/System/Settings/Rules/Item Saved/Rules` item.

   Store the rule definitions that handle version removal under the `/Sitecore/System/Settings/Rules/Version Removed/Rules` item.

   Store the rule definitions that validate items under the `/sitecore/system/Settings/Rules/Validation Rules/Rules` item.

   Store the rule definition items that handle conditional rendering under the `/Sitecore/System/Marketing Center/Personalization/Rules` item.

   Store the global conditional rendering rules under the `/Sitecore/System/Settings/Rules/Conditional Renderings/Global Rules` item.

   Store any other type of rule definition item under the appropriate descendant of the `/Sitecore/System/Settings/Rules` item.

2. In the **Content Editor**, insert a rule definition item that is based on the appropriate template.

   You can create rules based on the following templates:

Rule	Template
Conditional Rendering rule	`/sitecore/templates/System/Rules/Conditional Rendering Rule` data template
Content Editor Warning rule	`/sitecore/templates/System/Rules/Content Editor Warning Rule` data template
Insert Options rule	`/sitecore/templates/System/Rules/Insert Options Rule` data template
Validation rule	`/sitecore/templates/System/Rules/Validation Rule` data template
Others	`/sitecore/templates/System/Rules/Rule` data template

3. In the rule definition item, in the **Data** section, in the **Name** field, enter the name of the rule that should appear in user interfaces.
4. In the rule definition item, in the **Data** section, in the **Rule** field, select the conditional rendering conditions and actions, and then enter parameters for both the conditions and the actions.

Chapter 2

Conditional Rendering

This chapter describes how to implement a conditional rendering that allows runtime manipulation of presentation components. This chapter contains an example of a conditional rendering.

This chapter contains the following sections:

- Conditional Rendering
- Conditional Rendering Implementation
- Applying a Conditional Rendering

2.1 Conditional Rendering

The term conditional rendering refers to the Sitecore layout engine's ability to manipulate the presentation controls used to service an HTTP request by, for example,

- Including or excluding a rendering.
- Controlling its placeholder.
- Setting its data source and other properties.
- Processing multivariate conditions.
- And other logic.

Sitecore supports runtime personalization, whereby you can use a conditional rendering to personalize the content that a website visitor sees during a single session.

Important
You do not have to turn on the DMS to use runtime personalization.

2.2 Conditional Rendering Implementation

This section describes how to implement a conditional rendering.

2.2.1 Conditional Rendering Conditions

A conditional rendering condition contains logic that determines whether a condition is true and therefore whether to invoke a conditional rendering action. Layout details can reference conditional rendering rules, which refer to conditional rendering conditions.

Tip
You can use the conditions described in the section **Error! Reference source not found.** as conditional rendering conditions.

For information about implementing a conditional rendering condition, see the section *Conditions*.

2.2.2 Conditional Rendering Actions

A conditional rendering action contains logic that implements a conditional rendering feature, such as excluding the rendering or setting one of its properties. A conditional rendering action is an action like the ones described in the section Actions. Layout details reference conditional rendering rules, which in turn reference conditional rendering actions.

The conditional rendering actions are stored under the appropriate element in the `/sitecore/system/Settings/Rules/Definitions/Elements` folder.

How to Implement a Conditional Rendering Action

To implement a conditional rendering action:

1. In a Visual Studio project, create a class that inherits from the `Sitecore.Rules.Actions.RuleAction<T>` class, and implement the `Apply()` method.
2. In the **Content Editor**, navigate to the `/sitecore/system/Settings/Rules/Definitions/Elements/Conditional Renderings/` folder.
3. Create an action definition item based on the `/sitecore/templates/System/Rules/Action` data template.

   Name the conditional action definition item after the .NET class that implements the conditional rendering action.
4. In the conditional action definition item, in the **Data** section, in the **Text** field, enter the appropriate pattern.

   For more information about the **Text** field, see the section *Action Text*.
5. In the conditional action definition item, in the **Script** section, in the **Type** field, enter the signature of the .NET class that implements the conditional rendering action.
6. Configure the conditional rendering action in one or more conditional rendering rules as described in the section *Conditional Rendering Rules*.
7. Apply the conditional rendering rule as described in the section *Applying a Conditional* Rendering.

Example: Conditional Rendering Action to Add a Rendering

You can implement a conditional rendering action based on the following example that adds a rendering to a placeholder:

```
namespace Sitecore.Sharedsource.Rules.ConditionalRenderings
{
  using System;

  public class AddRenderingAction<T> : Sitecore.Rules.Actions.RuleAction<T>
    where  T: Sitecore.Rules.ConditionalRenderings.ConditionalRenderingsRuleContext
  {
    public string Placeholder
    {
      get;
      set;
    }

    public string RenderingID
    {
      get;
      set;
    }

    public override void Apply(T ruleContext)
    {
      Sitecore.Diagnostics.Assert.IsNotNullOrEmpty(
        this.RenderingID,
        "RenderingID");
      Sitecore.Data.Items.Item rendering =
        Sitecore.Context.Database.GetItem(this.RenderingID);
      Sitecore.Diagnostics.Assert.IsNotNull(rendering, "rendering");
      Sitecore.Layouts.RenderingReference rendRef =
        new Sitecore.Layouts.RenderingReference(rendering);

      if (!String.IsNullOrEmpty(this.Placeholder)
        && String.Compare(this.Placeholder, "specific", true) != 0)
      {
        rendRef.Placeholder = this.Placeholder;
      }

      ruleContext.References.Add(rendRef);
    }
  }
}
```

In the action definition item, enter the following text to allow the user to select a rendering and enter a placeholder key:

```
add the [RenderingID,Tree,root=/sitecore/layout/renderings,specific] rendering
  to the [Placeholder,Text,,specific] placeholder
```

2.2.3 Conditional Rendering Rules

Sitecore invokes conditional rendering rules to control conditional rendering logic. Conditional rendering rules associate one or more conditional rendering actions with one or more conditional rendering conditions, including parameter values for both actions and conditions.

Conditional rendering rules are based on the `System/Rules/Conditional Rendering Rule` data template.

2.3 Applying a Conditional Rendering

This section describes how to apply a conditional rendering.

2.3.1 How to Apply a Conditional Rendering

Configure the conditional rendering in the layout details of the standard values for data templates.

For more information about layout details, see the *Presentation Component Reference* manual.

2.3.2 Global Conditional Rendering Rules

For each rendering, in addition to evaluating conditional rendering rules selected in layout details, Sitecore evaluates the global conditional rendering rules defined under the `/Sitecore/System/Settings/Rules/Conditional Renderings/Global Rules` item.

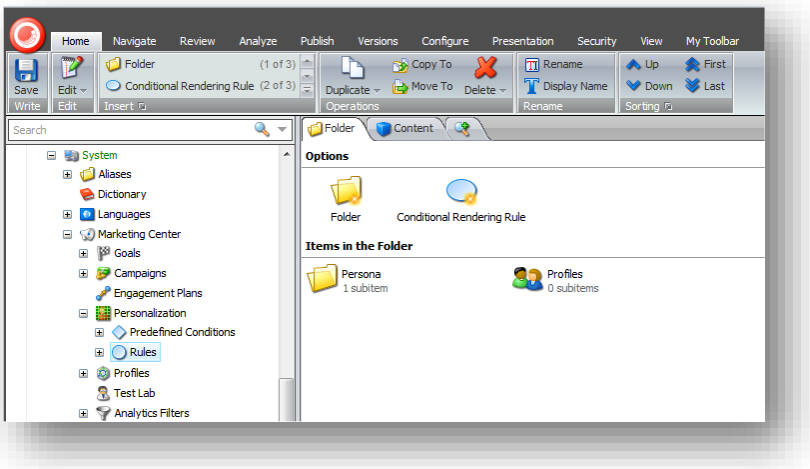
Important
Because they execute for each rendering and sublayout, it is important to optimize the performance of global conditional rendering rules, especially as the number of renderings specified in layout details increases.

To define a global conditional rendering rule, implement a conditional rendering rule under the `/Sitecore/System/Settings/Rules/Conditional Renderings/Global Rules` item as described in the section *How to Implement a Rule*, but using the `System/Rules/Conditional Rendering Rule` data template.

Creating Conditional Rules for an Individual Rendering

To create a conditional rendering rule for an individual rendering:

1. In the Content Editor, navigate to the `/sitecore/system/Marketing Center/Personalization/Rules` item.



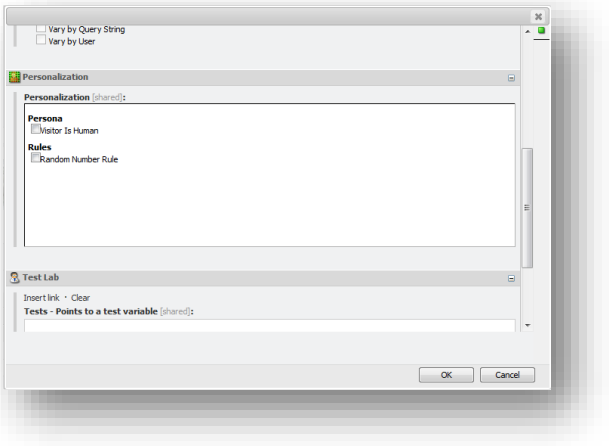
2. Click **Conditional Rendering Rule**, and give the conditional rendering rule a suitable name.

3. In the conditional rule item, in the **Rule** field, click **Edit Rule** and select the conditions and actions that you need in this rule.

To assign the conditional rendering rule to an individual rendering:

1. In the Content Editor, in the upper left corner, click **Sitecore**, **Application Options.**
2. In the **Application Options** dialog box, click the **View** tab, and in the **Control Properties** section, select the following checkboxes:
   - **Show the Personalization Section**
   - **Show the Test Lab Section**

   Now when you open the **Layout Details** dialog box and open the properties dialog box for an individual rendering, you see the following fields:



3. Select the personalization rule or conditional rendering that you want to use.

Chapter 3

Implementing a Conditional Rendering Rule

This chapter contains an example that shows you how to implement a conditional rendering.

It describes how to create the condition, action, and rule as well as how to apply the conditional rendering rule.

This chapter contains the following sections:

- Conditional Rendering Example

3.1 Conditional Rendering Example

This example explains how to create a conditional rendering rule.

In this example the computer generates a random number every time a user accesses a specific page. If this number matches a predefined number the user wins a prize.

Creating the Random Number Conditional Rendering Condition

To create this example, you must:

- Specify the upper limit for the random number.
- Specify the number that the random number must match to win.

The following example evaluates to `true` if a random number matches the specified parameter value:

```
namespace Sitecore.Sharedsource.Rules.Conditions
{
  using System;

  [UsedImplicitly]
  public class RandomNumberCondition<T> :
    Sitecore.Rules.Conditions.OperatorCondition<T>
    where T : Sitecore.Rules.RuleContext
  {
    private int upperLimit;

    public int MatchNumber
    {
      get;
      set;
    }

    public int UpperLimit
    {
      get
      {
        Sitecore.Diagnostics.Assert.IsTrue(this.upperLimit > 1, "upper limit");
        return this.upperLimit;
      }

      set
      {
        Sitecore.Diagnostics.Assert.IsTrue(value > 1, "value");
        this.upperLimit = value;
      }
    }

    protected override bool Execute(T ruleContext)
    {
      Sitecore.Diagnostics.Assert.ArgumentNotNull(ruleContext, "ruleContext");
      int random = new Random(DateTime.Now.Millisecond).Next(this.UpperLimit + 1);

      switch (this.GetOperator())
      {
        case Sitecore.Rules.Conditions.ConditionOperator.Equal:
          return random == this.MatchNumber;
        case Sitecore.Rules.Conditions.ConditionOperator.GreaterThanOrEqual:
          return random >= this.MatchNumber;
        case Sitecore.Rules.Conditions.ConditionOperator.GreaterThan:
          return random > this.MatchNumber;
        case Sitecore.Rules.Conditions.ConditionOperator.LessThanOrEqual:
          return random <= this.MatchNumber;
        case Sitecore.Rules.Conditions.ConditionOperator.LessThan:
          return random < this.MatchNumber;
```

```
            case Sitecore.Rules.Conditions.ConditionOperator.NotEqual:
              return random != this.MatchNumber;
          }

          return false;
        }
      }
    }
```

Now that you have created the random number conditional rendering condition, you must implement it.

Implementing the Condition

To implement the conditional rendering condition:

1. In the Visual Studio project, create the condition class by inheriting from:

Condition	Function
`Sitecore.Rules.Conditions.WhenCondition`,	For Boolean conditions
`Sitecore.Rules.Conditions.StringOperatorCondition`	For comparing string values
`Sitecore.Rules.Conditions.OperatorCondition`,	For comparing other types of values.

Tip
You can save time by duplicating an existing condition with similar functionality.

2. In the condition class, implement the `Execute()` method to indicate whether the condition is `true` or `false`.
3. In the **Content Editor**, select the item under which you want to store the condition definition item.

   The conditions must be stored under the appropriate descendant of the `sitecore/system/Settings/Rules/Definitions/Elements` item.
4. Insert a condition definition item based on the

   `/sitecore/templates/System/Rules/Condition` data template. Name the condition definition item after the .NET class that implements the condition.
5. In the condition definition item, in the **Data** section, in the **Text** field, enter some text as described in the section *Condition Text*.
6. In the condition definition item, in the **Script** section, in the **Type** field, enter the type signature of the .NET condition class.

You can now use this conditional rendering in the rules that you create. For more information about configuring conditions in rules, see the section *Conditional Rendering Rules*.

In the Content Editor, in the conditional rendering condition definition item, in the **Data** section, in the **Text** field, enter the following text to allow the user to set the `UpperLimit` and `MatchNumber` parameters of the conditional rendering condition, and to select an operator such as the equality operator ("="):

```
when a random number between 1 and [UpperLimit,,,upper limit]
[operatorid,Operator,,compares to] [MatchNumber,,,number]
```

The base class for the conditional rendering condition (`Sitecore.Rules.Conditions.OperatorCondition<T>`) defines the `OperatorId` property. The conditional rendering condition class (`Sitecore.Sharedsource.Rules.Conditions.RandomNumberCondition)` defines the `UpperLimit` and `MatchNumber` properties.

Note
To meet the strict requirements outlined for this example, you must hard-code the comparison logic instead of allowing the user to select an operator.

Save this conditional rendering condition in the `/sitecore/system/Settings/Rules/Definitions/Elements/Conditional Renderings` folder. This will ensure that it is tagged correctly.

Creating the Conditional Rendering Action

You can implement a conditional rendering action based on the following example that sets the `IsWinner` property of a rendering:

```
namespace Sitecore.Sharedsource.Rules.ConditionalRenderings
{
  using System;
  using Sitecore.Collections;

  [UsedImplicitly]
  public class SetWinner<T> : Sitecore.Rules.Actions.RuleAction<T>
    where T : Sitecore.Rules.ConditionalRenderings.ConditionalRenderingsRuleContext
  {
    public bool IsWinner
    {
      get;
      set;
    }

    public override void Apply(T ruleContext)
    {
      string key = "winner";

      if (String.IsNullOrEmpty(ruleContext.Reference.Settings.Parameters))
      {
        ruleContext.Reference.Settings.Parameters =
          key + "=" + this.IsWinner.ToString();
        return;
      }

      SafeDictionary<string> parameters = Sitecore.Web.WebUtil.ParseQueryString(
        ruleContext.Reference.Settings.Parameters);

      foreach (string check in parameters.Keys)
      {
        if (String.Compare(check, key, true) == 0)
        {
          key = check;
          break;
        }
      }

      parameters[key] = this.IsWinner.ToString();
      ruleContext.Reference.Settings.Parameters =
        Sitecore.Web.WebUtil.BuildQueryString(parameters, false);
    }
  }
}
```

In the **Text** field in the **Data** section of the conditional rendering action definition item, allow the user to specify a value for the `IsWinner` property if the random number matches:

```
set parameters to (true or false) [IsWinner,Text,,value]
```

The conditional rendering action class (`Sitecore.Sharedsource.Rules.ConditionalRenderings.SetWinner`) defines the `IsWinner` property.

Save this conditional rendering action in the `/sitecore/system/Settings/Rules/Definitions/Elements/Conditional Renderings` folder. This will ensure that it is tagged correctly.

Creating the Random Number Conditional Rendering Rule

You must insert a conditional rendering rule that generates a random number. You can call the rule definition item *Pick a Random Winner*.

1. To create a global conditional rendering rule, in the **Content Editor**, select the `/Sitecore/System/Settings/Rules/Conditional Renderings/Global Rules` item.

   To create an individual conditional rendering rule, in the **Content Editor**, select the `/Sitecore/System/Settings/ Marketing Center/Personalization/Rules` item.
2. On the **Home** tab, in the **Insert** group, click *Conditional Rendering Rule* to create a rule based on the `/sitecore/templates/System/Rules/Conditional Rendering Rule` data template.
3. In the rule definition item, in the **Data** section, in the **Rule** field, click **Edit Rule**.
4. In the **Rule Set Editor**, in the **Select the condition for the rule** field, select the *when a random number between 1 and upper limit compares to number* condition.
5. In the **Select the actions for the rule** field, select *set parameters to (true or false) value* action
6. In the **Rule description** field, select the parameters for the condition and the action that you want to implement in this conditional rendering rule.
7. In the conditional rendering rule definition item, in the **Data** section, in the **Rule** field, under Select the actions for the rule, select *user wins* to add that action to the list of actions that are executed if the specified condition evaluates to *True*.

   If you are creating a global conditional rendering rule, it is automatically assigned to every item.

   If you are creating an individual conditional rendering rule, you must assign this rule to the appropriate rendering.
8. To assign an individual conditional rendering rule, open the layout details of the standard values for the appropriate data templates and assign the rule.

Now when a visitor views this page a random number is generated. This number is then evaluated to see if it matches the predefined winner. If it does, you have a winner!