Sitecore CMS 6.0 or later

Include File Patching Facilities

Using include files to extend and patch configuration files in Sitecore

Introduction

The main purpose of the include file mechanism is to extend and patch Sitecore configuration files. Include files allow you to introduce new settings, properties, and configuration nodes, as well as override the values of existing settings, properties and configuration nodes.

There is a special XML syntax that allows you to perform common operations, such as:

- Inserting a node into a defined place.
- Modifying an attribute value.
- Removing a node.

This document describes this process in more detail.

General Concepts

XML Namespaces and Node Names

All the attributes and elements that are related to patching are located inside the following XML namespaces:

- `patch` – http://www.sitecore.net/xmlconfig/
- `set` – http://www.sitecore.net/xmlconfig/set/

You must declare the namespaces before you can use them in an include file. Normally, they are declared at the beginning of the file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
. . . . .
  . . . . .
</configuration>
```

In this scenario, you use the `patch:` and `set:` prefixes to access the nodes that control patching.

Useful terminology:

Term	Definition
Node	An XML document is a node tree. A node can be an element node, an attribute node, a text node, and so on.
Element	A logical document component which either begins with a start-tag and ends with a matching end-tag or only contains an empty-element tag, for example, `<element />`.
Attribute	A markup construct that consists of a name/value pair that exists within a start-tag or empty-element tag, for example, `<element attribute=”value” />`.

For more information about XML namespaces, see Wikipedia — http://en.wikipedia.org/wiki/XML_namespace.

The Include Directory and *.config files

Sitecore reads the configuration files in the following order to build up the actual configuration to use:

- The `web.config` file.
- All of the `*.config` files from the `/App_config/Include` folder. These files extend and modify the content of the `<sitecore />` node in the `web.config` file.

The files in the `/App_Config/Include` folder are applied sequentially by file name. Sitecore therefore applies the `AName.config` file before the `ZName.config` file. The order in which the include files are applied doesn’t really matter. However, if the same setting (or node) is modified in more than one include file, only the last modification is accepted. Furthermore, if you want to patch a node that is added from another include file, you must ensure that the include file with the patch is applied after the include file that adds the node.

Sitecore only takes `*.config` files into account. If you need to disable a particular `*.config` file, you can change its extension. We recommend that you change the file name to `*.config.disabled`.

When Sitecore reads the include files, it traverses the file system recursively. Sitecore first enumerates all the `*.config` files in the `/App_Config/Include` folder and then recursively enumerates all the sub-folders. You should take this order into account when you create sub-folders in the `/App_Config/Include` folder.

Processing Include Files

When Sitecore applies the changes from an include file, it recursively traverses all the elements in the include file starting from the root and tries to match each element to an element in the existing configuration.

Sitecore uses the element's name and all its attributes to match the nodes. To match the elements in the include files with the corresponding elements in the `web.config` file, your include files must specify in a unique way which nodes and/or attributes should be modified.

When Sitecore processes a particular element, there are two options:

- The element already exists and Sitecore modifies its content.
- The element does not exist and Sitecore inserts this new element.

Sitecore uses the following comparison algorithm:

1. Sitecore takes an element from an include file and extracts all its attributes with their values. When matching elements, all the nodes from the `set` and `patch` namespaces and the namespace declarations are ignored — for example, `<element xmlns:customNs=".." />`.
2. Sitecore checks whether an element at the same level with the same name and attribute values exists in the original document.
3. If more than one node with these constraints exists, Sitecore selects the first of these nodes. For instance, if we specify the `<processor />` element in the include file, only the first processor in the `web.config` file is selected.
4. If no node matches the constraints, Sitecore inserts the new element instead of patching an existing node.

Sitecore uses the attributes and their values as constraints to determine which specific node to change. If you do not specify enough attributes, the wrong element may be matched. For example, technically it is possible that a few event handlers have the same type attribute values but different method attribute values. If you only specify the type attribute, Sitecore will match the first event handler. On the other hand, if you specify redundant attribute values, Sitecore might not match any node if its attribute values were changed in the original document, for example in the `web.config` file or in the previous include file. Every time you create an include file, you must analyze which attributes you want to include and which you want to omit.

Example

This example should help you to understand the rules.

The following configuration is defined in the `web.config` file:

```
<configuration>
  <sitecore>
    <events timingLevel="custom">
      <event name="item:added" />
      <event name="item:removed">
        <handler type="ItemEventHandler" method="OnItemRemoved" />
        <handler type="ItemEventHandler" method="OnItemChanged" />
      </event>
    </events>
    <settings>
      <setting name="AliasesActive" value="true" />
      <setting name="AllowLogoutOfAllUsers" value="false" />
    </settings>
  </sitecore>
</configuration>
```

You want to remove the `OnItemChanged` event handler and change the value of the `AllowLogoutOfAllUsers` setting to `true`.

The content of the include file could be:

```
<configuration xmlns:patch=http://www.sitecore.net/xmlconfig/
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <events>
      <event name="item:removed">
        <handler type="ItemEventHandler" method="OnItemChanged" >
          <patch:delete />
        </handler>
      </event>
    </events>
    <settings>
      <setting name="AllowLogoutOfAllUsers" >
        <patch:attribute name="value" value="true" />
      </setting>
    </settings>
  </sitecore>
</configuration>
```

Ignore the patch specific nodes for a moment — they are described later in this document.

Notice how the event handler and the setting are specified.

Let's analyze the XML document.

- The `sitecore` element does not require any constrains because there is only one available `sitecore` element.
- The original events element contains the `timingLevel` attribute.

  This attribute is ignored because there is only one events node and ambiguity is impossible. Furthermore, you should not specify the `timingLevel` attribute because if you change the `timingLevel` value in the `web.config` file, Sitecore will not be able to match it in the future.
- There are several `event` elements and you should therefore use the `name` attribute to identify the exact element that you mean.
- For the `handler` element, the example uses both the `type` and `method` attributes.

  It is not enough to only use the `type` attribute because the `OnItemRemoved` handler would be matched in this case. Only using the `method` attribute would be sufficient, because there are no other handlers that contain this method.

  However, it goes against the logic — the `type` and `method` pairing represents a handler — and could cause problems if you introduce another handler with the same method attribute value.
- For the `settings` element, the example doesn't use any attributes, because only one such node exists.
- For the `setting` element, the example only uses the `name` attribute value.

  If you add the `value` attribute, it will work fine until you change the value in the `web.config` file or in any preceding include files.

The resulting configuration is:

```
<configuration>
  <sitecore>
    <events timingLevel="custom">
      <event name="item:added" />
      <event name="item:removed">
        <handler type="ItemEventHandler" method="OnItemRemoved" />
      </event>
    </events>
    <settings>
      <setting name="AliasesActive" value="true" />
      <setting name="AllowLogoutOfAllUsers" value="true" />
    </settings>
  </sitecore>
</configuration>
```

The Resulting Configuration (showconfig.aspx)

When Sitecore starts, it reads the configuration files and merges them to produce the resulting XML document. The resulting configuration document is stored in memory, and Sitecore uses it every time it needs some configuration values. Use the `showconfig.aspx` page to view the resulting XML file. The URL is: `http://hostName/sitecore/admin/showconfig.aspx`.

To check the actual configuration, we recommend that you look at the `showconfig.aspx` page rather than the `web.config` file and the stand-alone include files.

Troubleshooting

The main tool that you can use to troubleshoot the patching functionality is the `showconfig.aspx` page. It allows to see how each include file affected the whole configuration.

To troubleshoot the include facilities more effectively, the `patch:source` attribute was introduced in Sitecore 7.0. This attribute indicates the name of the include file that last altered the current node.

The main properties of the `patch:source` attribute are:

- It is added to nodes where one or more attributes have been added or modified by an include file.
- If several include files have modified attributes on a node, the `patch:source` attribute will only indicate the name of the last include file that applied changes to the node.
- The attribute is omitted if none of the include files modified the node.

This attribute is displayed in the `/sitecore/admin/showconfig.aspx` page and does not affect the actual configuration.

The Patch Namespace (http://www.sitecore.net/xmlconfig/)

Overview

The patch namespace contains two types of nodes:

- Attributes — `<elem patch:attribute="" … />`
- Elements — `<patch:element …. />`

To access these nodes, you must define the `http://www.sitecore.net/xmlconfig/` and `http://www.sitecore.net/xmlconfig/set/` namespaces. Although you can specify any aliases for the namespaces, `patch` and `set` are usually used:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
```

Attribute Nodes

Normally when you introduce a new element, it is appended at the end of its parent element. The attribute nodes of the patch namespace are used to determine the relative position of the inserted element. Normally these attributes are only used when you introduce new elements. These attributes are not designed for moving existing elements.

There are three attributes in this namespace:

- `patch:before`
- `patch:after`
- `patch:instead`

The attributes nodes are used to specify the relative position of the current node. The anchor nodes are specified as attribute values in XPath. The XPath is evaluated using the parent element as the context item. Each attribute has a short alias that you can use instead of its long name.

patch:before

Use this attribute to specify that the current element should be inserted before a sibling element.

Short alias: `b`

Syntax

```
<processor type="CustomC, CustomA" patch:before="*[@type='Sc, Sc]'" />
<processor type="CustomC, CustomA" patch:b="*[@type='Sc, Sc]'" />
```

Example

Original configuration:

```
<configuration>
  <sitecore>
```

```
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

Include file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set=”http://www.sitecore.net/xmlconfig/set/”>
  <sitecore>
    <element name="d" patch:before="*[@name=b]"/>
  </sitecore>
</configuration>
```

Resulting configuration:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="d"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

patch:after

Use this attribute to specify that the current element should be inserted after a sibling element.

Short alias: `a`

Syntax

```
<processor type=”CustomC, CustomA” patch:after=”*[@type=’Sc, Sc]’” />
<processor type=”CustomC, CustomA” patch:a=”*[@type=’Sc, Sc]’” />
```

Example

Original configuration:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

Include file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set=”http://www.sitecore.net/xmlconfig/set/”>
  <sitecore>
    <element name="d" patch:after="*[@name=b]"/>
  </sitecore>
</configuration>
```

Resulting configuration:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="d"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

patch:instead

Use this attribute to specify that the current element should be inserted instead of a sibling element. Unlike `patch:before` and `patch:after`, this attribute allows you to completely replace the original node or element.

Short alias: `i`

Syntax

```
<processor type="CustomC, CustomA" patch:instead="*[@type='Sc, Sc]'" />
<processor type="CustomC, CustomA" patch:i="*[@type='Sc, Sc]'" />
```

Example

Original configuration:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

Include file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <element name="d" patch:instead="*[@name=b]"/>
  </sitecore>
</configuration>
```

Resulting configuration:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="d"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

Element Nodes

Element nodes are tags from the `patch` namespace that are applied to their parents. This means that the `<patch:delete/>` and `<patch:attribute …/>` elements only affect the parent element which surrounds them. These nodes cannot contain child elements. If they contain child elements, they are ignored.

The patch namespace contains two element nodes:

- `patch:delete`
- `patch:attribute`

If you specify a non-existing element name, the element will be ignored. These nodes have aliases that you can use instead of their full names.

patch:delete

This element is a marker that indicates that the parent node should be removed. You should not specify any attributes or inner elements.

Short alias: `d`

Syntax

```
<elementToRemove>
  <patch:delete />
  <!-- or -->
  <patch:d />
</elementToRemove>
```

Example

Original configuration:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

Include file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set=”http://www.sitecore.net/xmlconfig/set/”>
  <sitecore>
    <element name="b">
      <patch:delete />
    </element>
  </sitecore>
</configuration>
```

Resulting configuration:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

In this example, to identify which exact element should be removed, you must specify the `name` attribute for the `element` node in the include file. If you omit the `name` attribute, the first element is removed — in this example, the one named `a`.

patch:attribute

This element is used to modify the value of an attribute of the parent element. Usually it is used to change the value of a setting or to change the type attribute. Changing the type attribute of pipeline processors or event handlers is often useful when you want to override parts of the default Sitecore functionality.

Short alias: `a`

Syntax

```
<elementToAlter>
  <patch:attribute name="parentAttributeName" value="newAttributeValue" />
<!–- or -->
  <patch:a name="parentAttributeName" value="newAttributeValue" />

</elementToAlter>
```

Alternative syntax:

```
<elementToAlter>

  <patch:attribute name="parentAttributeName">
    newAttributeValue
  </patch:attribute>
<!–- or -->
  <patch:a name="parentAttributeName">
    newAttributeValue
  </patch:a>
</elementToAlter>
```

Attribute	Description
`name`	The name of the parent attribute whose value should be changed.
`value`	The new value that should be assigned to the parent attribute.

In many cases, using `set: prefix` is more convenient than using `patch:attribute`.

For more information about `set:prefix` see *****

Example

Original configuration:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="b"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

Include file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <element name="b">
      <patch:attribute name="name" value="d" />
    </element>
  </sitecore>
</configuration>
```

Resulting configuration:

```
<configuration>
  <sitecore>
    <element name="a"/>
    <element name="d"/>
    <element name="c"/>
  </sitecore>
</configuration>
```

The Set Namespace (http://www.sitecore.net/xmlconfig/set/)

Overview

The set namespace is used to override attribute values. This namespace does not contain any predefined elements or attributes and can only be applied to attribute nodes.

The set namespace duplicates the functionality of the patch:attribute element. However, the set namespace is often more convenient to use because it allows you to set an attribute to a specific value, regardless of whether the attribute is already present in the node or not.

The set namespace prefix indicates that you want to override the current attribute value of the current element. It says *set the value of this attribute to the specified value, no matter what it was before. If this attribute is not there already, introduce it. You want to have this attribute with the specified value.*

When Sitecore compares elements and attributes to identify which node to patch, it ignores attributes which start with the set namespace prefix. You must therefore be careful when you use this namespace because a single element can contain more than one attribute with the same name. In this situation, you must specify the attribute and value without the set: prefix to identify the node to patch and then specify the attribute with the set: prefix to alter the attribute value. For example `<element attr="a" set:attr="b" />`.

Before you use the set namespace, you must define the `http://www.sitecore.net/xmlconfig/set/ XML` namespace.

For more information about node matching, see the section *Processing Include Files*.

Syntax

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/
xmlns:set=http://www.sitecore.net/xmlconfig/set/>
  <element attr1="value1" attr2="value2" set:attr3="newValue" />
</configuration>
```

If the original element node contains the attr3 attribute, the value is replaced by newValue. If the original element does not contain this attribute, it is introduced.

Examples

Original configuration:

```
<configuration>
  <sitecore>
    <element name="a" color="blue" />
    <element name="b" color="gray"/>
    <element name="c" color="red" />
    <element name="c" color="blue"/>
  </sitecore>
</configuration>
```

You want to change the color attribute for the b element.

The include file should identify the node itself and contain the color attribute with a new value. You must use the set prefix with the color attribute.

Include file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <element name="b" set:color="yellow"/>
  </sitecore>
</configuration>
```

Resulting configuration:

```
<configuration>
  <sitecore>
    <element name="a" color="blue" />
    <element name="b" color="yellow"/>
    <element name="c" color="red" />
    <element name="c" color="blue"/>
  </sitecore>
</configuration>
```

The `name="b"` attribute is used to identify the correct element — because there are a few. The set:`color="yellow"` attribute combined with the `set` namespace prefix is used to alter the value of the existing `color` attribute.

Let's do a trickier change.

You want to change the blue color to violet in the following element:

```
    <element name="c" color="blue" />
```

You can't use only the `name` or the `color` attribute because this will lead to ambiguity and Sitecore will select the first node that matches the specified attribute values.

To identify the node, you must use both the `name` and the `color` attributes. The element in the include file must also contain the `set` namespace with a new value for the `color` attribute.

Include file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <element name="c" color=”blue” set:color="violet"/>
  </sitecore>
</configuration>
```

Resulting configuration:

```
<configuration>
  <sitecore>
    <element name="a" color="blue" />
    <element name="b" color="gray"/>
    <element name="c" color="red" />
    <element name="c" color="violet"/>
  </sitecore>
</configuration>
```

The include file used the `name="c"` and the `color=”blue”` attributes to identify the node and used the `set:color="violet"` attribute to change the attribute value.

Multiple Approaches to Solving the Same Task

In the section about the `set` namespace, we explained that all the functionality of the `patch:attribute` can be achieved with the `set` namespace. Furthermore, we explained that the same task can be done in several different ways. Let's demonstrate this.

Example

You have the following configuration from Email Campaign Manager:

```
<configuration>
  <sitecore>
    <TypeResolver
type="Sitecore.Modules.EmailCampaign.Core.TypeResolver,Sitecore.EmailCampaign"
singleInstance="true" />
  </sitecore>
</configuration>
```

You want to override the default type resolver and set the custom type value to the type attribute.

There are several approaches to this task:

Approach #1: Using patch:delete & insert new

The most direct approach is to remove the whole `TypeResolver` node and specify a custom one. This can be done with the following include file:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/" >
  <sitecore>
    <TypeResolver>
      <patch:delete />
    </TypeResolver>
    <TypeResolver type="CustomTypeResolver, Custom" singleInstance="true" />
  </sitecore>
</configuration>
```

Approach #2: Using patch:instead

You can also use the `patch:instead` attribute to insert a custom `TypeResolver` element instead of the default one:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/" >
  <sitecore>
    <TypeResolver type="CustomTypeResolver, Custom" singleInstance="true"
patch:instead="TypeResolver" />
  </sitecore>
</configuration>
```

Approach #3: Using patch:attribute

The `patch:attribute` node allows us to modify the type attribute only and doesn't affect the other attributes, for example `singleInstance`. You can therefore also use it:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/" >
  <sitecore>
    <TypeResolver>
      <patch:attribute name="type">
        CustomTypeResolver, Custom
      </patch:attribute>
    </TypeResolver>
  </sitecore>
</configuration>
```

Approach #4: Using Set namespace

If you can achieve this result with the `patch:attribute` node, you can do the same with the `set` namespace. The include file could be:

```
<configuration xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <TypeResolver set:type="CustomTypeResolver, Custom" />
  </sitecore>
</configuration>
```

No matter which method you chose, the final configuration is the same:

```
<configuration>
  <sitecore>
    <TypeResolver type="CustomTypeResolver, Custom" singleInstance="true" />
  </sitecore>
</configuration>
```

Choosing a Method

You can use any of the methods described in this document to achieve your goal. However, we do not recommend all of these methods.

The first two approaches — using `patch:delete`/`patch:insert` or using `patch:instead` — completely override the `TypeResolver` element. They override the `singleInstance` attribute value and ignore its old value

If you change the original attribute value in the `web.config` file or any of the preceding include files, the actual value is not affected. The probability of this particular change is relatively low, so in this case it is a matter of taste.

However, there are places when it would be a mistake to follow this approach. For instance, if you want to override the database type:

```
<database id="master" singleInstance="true" type="Sitecore.Data.Database, Sitecore.Kernel">
. . . . . . . .
</database>
```

In this particular case, you must bring the whole node to the include file. Some preceding include files can affect the database settings, for example, tune caches sizes or specify custom engines and these changes will be ignored if you replace the entire node. You should therefore be careful when you decide which approach to use.

In the previous example, the last two approaches — using `patch:attribute` or the `set` namespace — are more elegant because they only affect the type attribute. The `set` namespace method requires less text and seems simpler. We therefore recommend any of the last two approaches.

Before you choose an approach, we recommend that you always consider the alternative approaches. In an ideal world, the approach you choose should affect only the attributes/elements that you are interested in and rely on a minimal number of other values.

Best Practices

Only Use CMS Patching Facilities to Alter CMS Configuration

The overwhelming majority of the Sitecore related configuration settings are located in the `<sitecore />` section of the application's configuration file. Sitecore incrementally applies `*.config` files to the `<sitecore />` section so any node inside that section could be easily overridden by any include file. It's highly recommended that you use the patching functionality to modify the resulting CMS configuration rather than modify it directly in the preinstalled configuration files, for example, the `web.config` or the `Sitecore.Analytics.config` file.

Using the include files has a number of benefits:

- It is much easier to track the places that modify CMS configuration and revert them if need.

  If the configuration is modified through the custom `*.config` files, you just need to rename those files to get back to the default CMS configuration.

- It is much easier to upgrade your CMS installation when you deploy your custom settings into a plain CMS installation.

- It is much easier to detect and troubleshoot customizations which affect particular pieces of CMS functionality.

- There is less chance that you will break the functionality of CMS modules.

  For instance, some modules deploy their own custom `*.config` files to extend CMS functionality with custom configurations. As we know, patching commands often require exact node values to work properly, for example, type attribute values. If you modify the default values directly in the `web.config` file, the patching functionality will not work as expected and the module could get broken. Of course, using patching facilities doesn't prevent these issues — the value could be altered by a preceding include file — but it is much easier to identify them if you only use `*.config` files to alter the configuration.

Because of the benefits it's recommended that you use include files to alter the configuration in the following situations:

- When modifying the default CMS or modules setting values, object types, and so on.

- When introducing new processors, settings, providers, and so on.

We recommend that you name custom `*.config` files with the name of your company/project and also that you include a short description of the changes that the file applies.

For example, if you want to alter the title on the Sitecore login page, the include file could be:

Include file name: `MyCompany.WelcomeTitle.config`

Include file content:

```
        <configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
          <sitecore>
            <settings>
              <setting name="WelcomeTitle" set:value="Welcome to MyCompany" />
            </settings>
          </sitecore>
        </configuration>
```

Add New Processors to Pipelines rather than Modify the Existing Ones

If you need to modify the behavior in a pipeline, it is often enough to just alter the value of some of the pipeline arguments. In these cases, it's preferable to introduce a custom processor rather than rewrite the existing one.

This approach has some positive effects:

- It's much easier to upgrade your solution, because there are less chances that something will get broken.
- There is greater compatibility with other components which rewrite the same processor.

  Sometimes modules replace processors with custom ones. If you rewrite the processor, it is likely that the module will break.

For example, if the context item wasn't resolved by the default item resolver, you may need to resolve it in a special way. It would be nice to just add a custom item resolver immediately after the default one and have this do the work if the context item is null — the item was not resolved:

Web.config file content:

```
<httpRequestBegin>
 ...
  <processor type="Sitecore.Pipelines.HttpRequest.ItemResolver, Sitecore.Kernel" />
  ...
</httpRequestBegin
```

Include file name:

`\App_Config\Include\MyCompany.ItemResolver.config`

Include file content:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
  <sitecore>
    <pipelines>
      <httpRequestBegin>
        <processor type="MyCompany.ItemResolver, MyCompany"
patch:after="*[@type='Sitecore.Pipelines.HttpRequest.ItemResolver, Sitecore.Kernel']" />
      </httpRequestBegin>
    </pipelines>
  </sitecore>
</configuration>
```

Always Add New Provider Rather than Modify the Type of the Default

If you need to extend the logic of an existing provider, for example, `Sitecore.Security.Authentication.FormsAuthenticationProvider`, we strongly recommended that you add a new provider with the altered logic rather than replace the type of the default one. This increases the supportability of the product and allows you to quickly switch to the default provider.

To introduce new provider, we recommend that you use this approach:

Initial configuration:

```
<configuration>
  <sitecore>
    <authentication defaultProvider="forms">
      <providers>
        <clear/>
        <add name="forms" type="Sitecore.Security.Authentication.FormsAuthenticationProvider,
Sitecore.Kernel"/>
```

```
      </providers>
    </authentication>
  </sitecore>
</configuration>
```

Include file content:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/"
xmlns:set="http://www.sitecore.net/xmlconfig/set/">
  <sitecore>
    <authentication set:defaultProvider="custom">
      <providers>
        <add name="custom" type="CustomType, CustomAssembly"/>
      </providers>
    </authentication>
  </sitecore>
</configuration>
```

This approach will add one more provider and force Sitecore to use this provider rather than the default one.