Sitecore CMS 6.0 or later

Data Definition Cookbook

Tips and Techniques for CMS Administrators, Architects, and Developers

Table of Contents

Chapter 1

Introduction

This Sitecore Data Definition Cookbook provides tips and techniques for CMS Administrators, Architects, and Developers analyzing and implementing the data infrastructure components of Sitecore solutions. This document includes information about data templates, branch templates, command templates, insert options, proxies, and aliases.[1]

This document contains the following chapters:

- Chapter 1 – Introduction
- Chapter 2 – Data Templates
- Chapter 3 – Branch Templates
- Chapter 4 – Command Templates
- Chapter 5 – Insert Options
- Chapter 6 – Aliases and Proxies

[1] For more information about the topics described in this cookbook, see the Data Definition Reference guide at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

Chapter 2

Data Templates

This chapter provides procedures for working with data templates, including steps to access the Template Manager, view the properties of a data template, creating and updating data templates, and working with fields and standard values.

This chapter contains the following sections:

- How to Access the Template Manager
- Investigate Data Template Properties
- Working with Data Templates
- Working with Fields
- Field Standard Values

2.1 How to Access the Template Manager

You can use the Template Manager to configure data templates.

To access the Template Manager, in the Sitecore Desktop, click the Sitecore button, and then click Template Manager. The Template Manager appears within the Sitecore Desktop.

Note
You can configure data templates using the Template Manager, or the Content Editor. The functional difference between these two applications is the root item of the content tree shown in the user interface. The root item of the content tree in the Template Manager is `/Sitecore/Templates`. The root item of the content tree in the Content Editor is `/Sitecore`. To see the `/Sitecore/Templates` item in the Content Editor, you may need view hidden items.[2]

[2] For instructions to view hidden items, see the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

2.2 Investigate Data Template Properties

This section provides procedures to determine the data template associated with an item and view the properties of a data template.

2.2.1 How to Determine the Data Template Associated with an Item

To determine the data template associated with an item:

1. In the Content Editor, select the item.
2. In the Quick Info section, investigate the Template property.

2.2.2 How to Determine the Base Template(s) of a Data Template

To determine the base template(s) associated with a data template:

1. In the Template Manager or the Content Editor, select the data template definition item. The Template Builder appears.
2. In the Template Builder, click the Inheritance tab. A report listing the base templates associated with the data template appears, including the inheritance hierarchy and the structure of sections and fields in those templates.

2.3 Working with Data Templates

This section provides procedures for working with data templates.

2.3.1 How to Edit the Data Template Associated with an Item

To edit the data template associated with an item:

1. In the Content Editor, select the item.
2. Click the Configure tab.
3. In the Template group, click the Edit command. The Template Manager appears with the data template associated with the item selected.

2.3.2 How to Create a New Data Template

To create a new data template:

1. In the Template Manager or the Content Editor, within `/Sitecore/Templates`, insert any required project-specific folders using the `/System/Templates/Template Folder` data template.
2. Under the appropriate project-specific folder, insert a new data template definition item using the `/Branches/System/Templates/New Template` branch template. The new data template dialog appears.
3. In the new data template dialog, for Name, enter the name of the new data template.
4. For Base template, select a base data template, or select the default `/System/Templates/Standard template` base data template.[3]
5. Click Next.
6. Select the project-specific folder that will contain the new data template, and then click Next.
7. Click Finish.

2.3.3 How to Configure the Icon for a Data Template

You can configure an icon for a data template to control the default icon shown for all items based on the data template.[4]

To configure the icon for a data template:

1. In the Template Manager or the Content Editor, select the data template definition item.
2. Click the Configure tab.
3. In the Appearance group, click the Icon command. The icon selection dialog appears.
4. In the icon selection dialog, select an icon.

Note
Unlike most data template properties, configure the icon for a template in the template definition item, not in the template's standard values.

[3] For more information about the standard template, see the Data Definition Reference manual at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.
[4] For more information about icons, see the Client Configuration Reference manual and the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

2.3.4 How to Configure Base Templates for a Data Template

To configure the base templates for a data template:

1. In the Template Manager or the Content Editor, select the data template definition item. The Template Builder appears.
2. In the Template Builder, click the Builder Options tab.
3. In the Template group, click the Base Templates command. The Base Templates dialog appears.
4. In the Base Templates dialog, select base templates, and then click OK.

Template Inheritance Example: Configure a Template Hierarchy

Consider the following simplified requirements:

- All content items contain a field named Title.
- Some types of content items contain Title and a field named Body.
- NewsArticle content items contain Title, Body, and another field named ReleaseDate.
- Product content items contain Title, Body, and another field named Department.
- Both NewsArticle and Product content items contain another field named Image.
- JobDescription content items contain Title, Body, and Department, but do not contain Image.

The following outline provides one of several possible data template inheritance hierarchies to meet these requirements. Unless otherwise specified, all data templates inherit from the standard data template.

1. Create the Page data template containing the Title field.
2. Create the ContentPage data template inheriting from Page and containing the Body field.
3. Create the HasDepartment data template containing the Department field.
4. Create the HasImage data template containing the Image field.
5. Create the ImageContentPage data template inheriting from both ContentPage and HasImage.
6. Create the NewsArticle data template inheriting from ImageContentPage and defining the ReleaseDate field.
7. Create the Product data template inheriting from both ImageContentPage and HasDepartment.
8. Create the JobDescription data template inheriting from both ContentPage and HasDepartment.

Note
Do not use template inheritance for fields that have different properties in different data templates. For example, if the Image field in NewsArticle and Product should retrieve images from different locations in the media library, define Image fields in each data template instead of inheriting from a common data template containing the Image field.

Tip
It is convenient, but not necessary, to create base templates before creating the data templates that inherit from them, even if you do not fully define the sections and fields of the base templates before creating the inheriting templates.

2.3.5 How to Add a Section to a Data Template

To add a data template section to a data template:

1. In the Template Manager or the Content Editor, select the data template definition item. The Template Builder appears.
2. In the Template Builder, in the first field containing the text Add a new section, enter the name of the new data template section.
3. In the Write group, click the Save command.

Important
The standard template uses the section names Advanced, Appearance, Help, Layout, Lifetime, Insert Options, Publishing, Security, Statistics, Tasks, Validators, and Workflow. Avoid using these section names in your data templates.

How to Configure the Icon for a Data Template Section

To configure the icon that appears in the Content Editor for a data template section:

1. In the Template Manager or the Content Editor, expand the data template definition item.
2. In the content tree, select the data template section definition item.
3. Click the Configure tab.
4. In the Appearance group, click the Icon command. The icon selection dialog appears.
5. In the icon selection dialog, select an icon.

2.3.6 How to Add a Field to a Data Template

To add a data template field to a data template:

1. In the Template Manager or the Content Editor, select the data template definition item. The Template Builder appears.
2. In the Template Builder, add at least one data template section.
3. In the Template Builder, under the data template section, in the first field containing the text Add a new field, enter the name of the new data template field.
4. In the Type column, select a type for the new data template field.
5. In the Write group, click the Save command.

2.3.7 How to Associate an Item with a Different Data Template

You may associate an existing item with a different data template. For example, you may need to change the data template associated with an item to turn a general page into a news article. For each field value in the item, if the new data template contains a field with the same ID or name as the original data template, the item retains that field value.

To associate an item with a different data template:

1. In the Content Editor, select the item.
2. Click the Configure tab.
3. In the Template group, click the Change command. The Select the Template dialog appears.
4. In the Select the Template dialog, select the new data template for the item, and then click Next.

2.3.8 How to Validate Data Template Field Values

To validate data entered into data template fields, configure validation rules in the data template field definition items.[5]

To configure validation rules for a data template field:

1. In the Template Manager or the Content Editor, edit the data template field definition item.
2. For the fields in the Validation Rules section, select validation rules.

2.3.9 How to Validate Items Based on a Data Template

To validate data entered into all items based on a data template, configure validation rules in the template's standard values.

To configure default validation for all items based on a data template:

1. In the Template Manager or the Content Editor, edit the data template definition item. The Template Builder appears.
2. Click the Builder Options tab.
3. In the Template group, click the Standard Values command. If it does not exist, Sitecore inserts the standard values item under the data template definition item.
4. Show the standard template fields.[6]
5. For the fields in the Validation Rules section, select validation rules.
6. Hide the standard template fields.

[5] For more information about validation, see the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.
[6] For instructions to show and hide the standard template fields, see the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

2.4 Working with Fields

This section provides procedures to configure data template field properties.

2.4.1 How to Configure Field Properties

To configure the properties of a data template field:

1. In the Template Manager or the Content Editor, expand the data template definition item and the data template section definition item containing the field, and then select the data template field definition item.
2. In the data template field definition item, enter field properties.

2.4.2 How to Control the List of Items in a Selection Field

The Source property of data template fields supports three techniques for controlling the items that appear in the selection list for fields that allow the user to select zero or more items.

Note
If you set the Source property of a data template field to a Sitecore query or Treelist parameters as described in the following sections, you may receive a prompt indicating that the Source field in the data template field definition item contains one or more broken links. You can ignore this warning and save the change to the data template field definition item.

Root Item

You can set the Source property of a Droplink, Droplist, Droptree, File, Grouped Droplink, Grouped Droplist, Image, Multilist, Treelist, or TreelistEx field to a root item. For single-level lists such as Checklist, Droplink, Droplist, or Multilist, users can select a child of the root item. For grouped lists such as Grouped Droplink and Grouped Droplist, users can select a grandchild of the root item. For tree lists such as Droptree, File, Image, Treelist, and TreelistEx, users can select a descendant of the root item. For example, to allow selection from beneath the `/Sitecore/Content/Home` item, set the Source property as follows:

```
/Sitecore/Content/Home
```

Sitecore Query

You can set the Source property of a Checklist, Droplink, Droplist, Grouped Droplist, Grouped Droplink, or Multilist field to a Sitecore query using the `query:` prefix.[7] For example, to allow selection of children of the home item that are based on the data template named Section:

```
query:/sitecore/content/home/*[@@templatename='Section']
```

Treelist Parameters

You can set the Source property of a Treelist or TreelistEx field to a number of URL-escaped key=value parameters separated by ampersand characters (“`&`”). For example, to allow users to select descendants of the home item that are based on either the data template named Section or the data template named Page:

```
DataSource=/Sitecore/Content/Home&IncludeTemplatesForSelection=section,page
```

You can specify the following parameters:

[7] For more information about Sitecore query, see http://sdn.sitecore.net/Reference/Using%20Sitecore%20Query.aspx and the Data Definition Reference manual at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

- **`DataSource`**: Root item. For more information, see the previous section Root Item.
- **`DatabaseName`**: The name of the database containing `DataSource`.
- **`IncludeTemplatesForSelection`**: Users can only select items based on this comma-separated list of data template names.
- **`ExcludeTemplatesForSelection`**: Users cannot select items based on this comma-separated list of data template names.
- **`IncludeTemplatesForDisplay`**: Users can navigate items based on this comma-separated list of data template name and IDs.
- **`ExcludeTemplatesForDisplay`**: Users cannot navigate items based on this comma-separated list of data template names and IDs.
- **`IncludeItemsForDisplay`**: Users can navigate items based on this comma-separated list of item names and IDs.
- **`ExcludeItemsForDisplay`**: Users cannot navigate items based on this comma-separated list of item names and IDs.
- **`AllowMultipleSelection`**: If `yes`, users can select the same item more than once.

2.4.3 How to Specify a Root Item for a File or Image Field

To specify the root item for a data template field of type File or Image, set the Source property in the field definition item as described in the section Root Item. For example, to allow users of an Image field to select a descendant of `/Sitecore/Media Library/Images`, set the Source property in the field definition item as follows:

```
/Sitecore/Media Library/Images
```

2.4.4 How to Specify a Default Folder for an Image Field

To specify the default location in the content tree for a data template field of type Image, set the Source property in the data template field definition item to a path starting with the tilde character ("~"). The selection list will default to this location, but users will be able to navigate to the root of the media library. For example, to default the location of the selector for an Image field to `/Sitecore/Media Library/Images`, set the Source property in the field definition item as follows:

```
~/Sitecore/Media Library/Images
```

Note
You cannot specify both a root item and a default folder for a data template field of type File or Image.

2.4.5 How to Configure Features Available in a Rich Text Editor Field

Rich Text Editor (RTE) profiles control features available in Rich Text Editor fields.[8] Set the Source property of an RTE field definition item to the path to an RTE profile definition item within `/Sitecore/System/Settings/Html Editor Profiles` in the Core database.

[8] For more information about Rich Text Editor Profiles, see the Client Configuration Reference manual and the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

2.5 Field Standard Values

This section provides procedures to working with field standard values.

2.5.1 How to Configure Standard Values for Fields

To configure standard values for the fields of a data template:

1. In the Template Manager or the Content Editor, select the data template definition item. The Template Builder appears.
2. Click the Builder Options tab.
3. In the Template group, click the Standard Values command. If it does not exist, Sitecore inserts the standard values item under the data template definition item.
4. In the standard values item, enter standard values for fields, and use the ribbon to apply standard values for properties stored in the fields of the standard template, such as insert options, layout details, and initial workflow.

2.5.2 How to Assign Standard Layout Details

To assign default layout details for all items based on a data template:[9]

1. In the Template Manager or the Content Editor, configure standard values for the data template.
2. Click the Presentation tab.
3. In the Layout group, click the Design command or the Details command.
4. Assign layout details.

2.5.3 How to Assign Standard Insert Options

For more information about insert options, see Chapter 5, Insert Options.

To assign default insert options for all items based on a data template:

1. In the Template Manager or the Content Editor, configure standard values for the data template.
2. Click the Configure tab.
3. In the Insert Options group, click the Assign command.
4. Assign insert options.

2.5.4 How to Assign a Default Workflow

To assign a default workflow for all items based on a data template:

1. In the Template Manager or the Content Editor, configure standard values for the data template.
2. Click the Review tab.
3. In the Workflow group, click the Initial command.
4. Select a default workflow for all items based on the data template.

[9] For more information about layout details, see the Presentation Component Reference Guide at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

2.5.5 How to Reset a Field to its Standard Value

To reset a field in an item to its standard value:

1. In the Content Editor, select the item.
2. If the field is defined in the standard template, show the standard template fields.
3. Click the Versions tab.
4. In the Fields group, click the Reset command. The Reset Fields dialog appears, showing item field values on the left and standard values on the right.
5. In the Reset Fields dialog, select the checkboxes for the field(s) to reset to their standard values, and then click Reset. Sitecore resets the selected fields to their standard values.
6. If the standard template fields are visible, hide the standard template fields.

2.5.6 How to Reset Layout Details to Standard Values

Sitecore provides a shortcut to reset layout details to their standard value.

To reset layout details to their standard value:

1. In the Content Editor, select the item.
2. Click the Presentation tab.
3. In the Layout group, click the Reset command, and then acknowledge the prompt. Sitecore resets layout details for the item to those defined in its template's standard values.

2.5.7 How to Reset Insert Options to Standard Values

Sitecore provides a shortcut to reset insert options to their standard value.

To reset inert options to their standard value:

1. In the Content Editor, select the item.
2. Click the Configure tab.
3. In the Insert Options group, click the Reset command, and then acknowledge the prompt. Sitecore resets insert options for the item to those defined in its template's standard values.

2.5.8 How to Reset a Field to its Standard Value Using .NET APIs

You can use the `Sitecore.Data.Fields.Field.Reset()` method to reset a field to its standard value. For example, to reset layout details for the context item to their standard value, use code such as the following:

```
Sitecore.Data.Items.Item item = Sitecore.Context.Item;
Sitecore.Data.Fields.Field field = item.Fields[Sitecore.FieldIDs.LayoutField];
item.Editing.BeginEdit();
field.Reset();
item.Editing.EndEdit();
```

Chapter 3

Branch Templates

This chapter provides procedures for configuring branch templates, which define prototypes for item structures that users can replicate when inserting new items. This chapter includes instructions to create a new branch template, to duplicate existing items to create a new branch template, to configure access rights for items inserted using a branch template, and for configuring access rights to control which CMS users can use different branch templates.

This chapter contains the following sections:

- Branch Template Overview
- How to Create a Branch Template
- How to Create a Branch Template for a Data Template
- How to Create a Branch Template by Duplicating Existing Items
- How to Configure the Icon for a Branch Template
- How to Configure Explicit Access Rights for Items Inserted Using a Branch Template
- How to Control Which Accounts Can Insert Items Using a Branch Template

3.1 Branch Template Overview

A branch template consists of a branch template definition item, which can contain a single item, a hierarchy of items, or multiple hierarchies of items. When a user invokes a branch template, Sitecore duplicates the item(s) beneath the branch template definition item, including any field values, and then performs token substitution on item names and field values as described in the following section How to Create a Branch Template. You can assign branch templates along with data templates and command templates as described in Chapter 5, Insert Options.

Note
As described in the following sections, you can insert a new branch template definition item using the `/System/Branches/Branch` data template or the `/Branches/System/Branch/New Branch` command template. If you use the command template, Sitecore creates a branch template definition item named after the data template you specify, and creates an item named `$name` based on that data template within the branch template definition item.

You can enter values including the following tokens in the fields of each item within the branch template definition item. Sitecore will expand these tokens in field values in each of the items inserted using the branch template.

- **`$name`**: The name entered by the user when inserting the item.
- **`$id`**: The ID of the item.
- **`$parentid`**: The ID of the parent of the item.
- **`$parentname`**: The name of the parent of the item.
- **`$date`**: The system date (`yyyyMMdd`).
- **`$time`**: The system time (`HHmmss`).
- **`$now`**: The date and time (`yyyyMMddTHHmmss`).

Note
You do not need to create a branch template for every data template. Use branch templates when you want to insert multiple items, assign access rights to inserted items, or control which accounts can insert items using the branch template.

Note
You can create any number of items within a branch template definition item, and each item can have any number of descendant items. When a user creates an item using the branch template, Sitecore replicates all descendants of the branch template definition item before expanding tokens.

Important
Unlike standard values, which Sitecore stores in a single standard values item for each data template and applies to all items based on that template that do not have values for those fields, Sitecore duplicates field values within branch template definition items to each item created using that branch template. To avoid data duplication, use standard values instead of field values in branch templates.

3.2 How to Create a Branch Template

To create a new branch template:

1. In the Template Manager or the Content Editor, within `/Sitecore/Tempaltes/Branches`, insert any required project-specific folders using the `/System/Branches/Branch Folder` data template.
2. Under the appropriate project-specific folder, insert a new branch template definition item using the `/System/Branches/Branch` data template.
3. Within the branch template definition item, insert one or more items or hierarchies of items.

3.3 How to Create a Branch Template for a Data Template

You can insert a new branch template containing a single item based on a specific data template.

To create a new branch template for a data template:

1. In the Template Manager or the Content Editor, within `/Sitecore/Tempaltes/Branches`, insert any required project-specific folders using the `/System/Branches/Branch Folder` data template.
2. Under the appropriate project-specific folder, insert a new branch template definition item using the `/Branches/System/Branch/New Branch` command template. The Create a New Branch dialog appears.
3. In the Create a New Branch dialog, select the data template, and then click Create. Sitecore creates a new branch template definition item containing a single item named `$name` based on the selected data template.

Note
By default, insert options for branch folders include the `/Branches/System/Branch/New Branch` command template, but not the `/System/Branches/Branch` data template.

3.4 How to Create a Branch Template by Duplicating Existing Items

You can create a branch template by duplicating one or more existing items or branches of items.

To create a branch template by duplicating one or more existing items:

1. In the Content Editor or the Template Manager, insert a branch template definition item using the `/System/Branches/Branch` data template.
2. In the Content Editor, show hidden items.[10]
3. Select the item to duplicate to populate the new branch template.
4. Click the Home tab.
5. In the Operations group, click the Copy To command. The Copy Item To dialog appears.
6. In the Copy Item To dialog, select the branch template definition item, and then click Copy. Sitecore copies the item or hierarchy of items to create a new item or hierarchy of items under the branch template definition item.
7. Hide hidden items if desired.

[10] For instructions to show hidden items, see the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

3.5 How to Configure the Icon for a Branch Template

Items created for a branch template display the icon associated with their data templates. You can configure the icon for a branch template to control the icon that appears for that branch template in insert options.

To configure the icon for a branch template:

1. In the Template Manager or the Content Editor, select the branch template definition item.
2. Click the Configure tab.
3. In the Appearance group, click the Icon command. The icon selection dialog appears.
4. In the icon selection dialog, select an icon.

3.6 How to Configure Explicit Access Rights for Items Inserted Using a Branch Template

By default, when you insert item using a branch template, those items inherit access rights from the parent. You can configure access rights for items in branch templates instead of inheriting access rights.

To configure access rights for items inserted using a branch template, configure access rights on the items within the branch template definition item. Sitecore will copy these access rights to items inserted using the branch template.

3.7 How to Control Which Accounts Can Insert Items Using a Branch Template

All users that have the `item:create` access right can insert items using any branch template available through insert options. You can configure access rights in branch template definition items to restrict which of these users can insert items using specific branch templates.

To control which users can insert items using a branch template, configure access rights on the branch template definition item. Users that do not have the `item:read` access right to a branch template definition item cannot insert items using that branch template.

Chapter 4

Command Templates

This chapter provides procedures for configuring command templates, which contain logic, most commonly to insert one or more new items. You can create a command template that displays no user interface, a command template that uses a JavaScript prompt to collect an item name, or a command template that displays an ASP.NET or Sitecore Sheer user interface. You can assign command templates along with data templates and branch templates using insert options.

This chapter contains the following section:

- How to Create a Command Template
- Command Template Example: Copy Insert Options from the Parent Item
- Command Template Example: Display a User Interface

4.1 How to Create a Command Template

To create a command template:

1. Create a class that inherits from `Sitecore.Shell.Framework.Commands.Command`, and override the `Execute()` method. You can use the following code sample:

```
namespace Namespace.Shell.Framework.Commands
{
  public class ClassName : Sitecore.Shell.Framework.Commands.Command
  {
    public override void Execute(Sitecore.Shell.Framework.Commands.CommandContext
      context)
    {
    }
  }
}
```

2. Add a `/configuration/command` element to the file `/App_Config/Commands.config` mapping the command code in the `name` attributre to the class signature in the `type` attribute. For example:

```
<command name="namespace:category:command"
type="Namespace.Shell.Framework.Commands.ClassName,Assembly" />
```

3. In the Template Manager or the Content Editor, within `/Sitecore/Templates`, insert any required project-specific folders using the `/Templates/Common/Folder` data template.
4. Under the appropriate project-specific folder within `/Sitecore/Templates`, insert a command template definition item using the `/System/Branches/Command Template` data template.
5. In the command template definition item, in the Data section, for the Command field, enter the command code. For example:

```
namespace:category:command(id=$ParentID)
```

Note
The parameter `id=$ParentID` sets the ID passed to the command to the item under which the user is inserting a new item. This causes Sitecore to pass the correct item ID to the command template when the user has selected an item, but has right-clicked another item before activating the command template. Without this parameter, Sitecore would pass the ID of the selected item rather than the item that the user right-clicked.

6. To assign insert options to other items to allow use of the new command template, see the section How to Assign Insert Options to a Data Template or an Individual Item.

4.2 Command Template Example: Copy Insert Options from the Parent Item

Consider the requirement that when a user creates a folder, Sitecore should copy the insert options from the parent item to the new folder.

The following command template code meets this requirement:

```
namespace Namespace.Shell.Framework.Commands
{
  public class NewAssignedFolder : Sitecore.Shell.Framework.Commands.Command
  {
    public override void Execute(Sitecore.Shell.Framework.Commands.CommandContext
      context)
    {
      if (context.Items.Length == 1)
      {
        Sitecore.Data.Items.Item item = context.Items[0];
        System.Collections.Specialized.NameValueCollection parameters =
          new System.Collections.Specialized.NameValueCollection();
        parameters["id"] = item.ID.ToString();
        parameters["language"] = item.Language.ToString();
        parameters["database"] = item.Database.Name;
        Sitecore.Context.ClientPage.Start(this, "Run", parameters);
      }
    }

    protected void Run(Sitecore.Web.UI.Sheer.ClientPipelineArgs args)
    {
      if (args.IsPostBack)
      {
        if ((!String.IsNullOrEmpty(args.Result)) && args.Result != "undefined")
        {
          Sitecore.Data.Database db = Sitecore.Configuration.Factory.GetDatabase(
            args.Parameters["database"]);
          Sitecore.Data.Items.Item parent = db.GetItem(args.Parameters["id"],
            Sitecore.Globalization.Language.Parse(args.Parameters["language"]));
          Sitecore.Data.Items.TemplateItem template =
            Sitecore.Context.ContentDatabase.Templates[Sitecore.TemplateIDs.Folder];
          Sitecore.Data.Items.Item item = parent.Add(args.Result, template);

          if (!(String.IsNullOrEmpty(parent[Sitecore.FieldIDs.Branches])
            && String.IsNullOrEmpty(parent["__Insert Rules"])))
          {
            item.Editing.BeginEdit();

            if (!String.IsNullOrEmpty(parent[Sitecore.FieldIDs.Branches]))
            {
              item.Fields[Sitecore.FieldIDs.Branches].Value =
                parent[Sitecore.FieldIDs.Branches];
            }

            if (!String.IsNullOrEmpty(item["__Insert Rules"]))
            {
              item.Fields["__Insert Rules"].Value = parent["__Insert Rules"];
            }

            item.Editing.EndEdit();
          }

          Sitecore.Context.ClientPage.SendMessage(this,
            "item:load(id=" + item.ID.ToString() + ")");
        }
      }
      else
      {
        Sitecore.Context.ClientPage.ClientResponse.Input(
          "Enter the name of the new folder:",
          Sitecore.Globalization.Translate.Text("New folder"),
          Sitecore.Configuration.Settings.ItemNameValidation,
          "'$Input' is not a valid name.",
```

```
                  100);
                args.WaitForPostBack();
              }
            }
          }
        }
```

When the user activates this command template, Sitecore invokes the `Execute()` method, which prepares information about the processing context and passes it to the `Run()` method. The `Run()` method prompts the user for an item name, then creates a folder by that name, and copies insert options from the parent item to the new item.

4.3 Command Template Example: Display a User Interface

You can use a command template that displays a user interface in which the user can select options when inserting items.

Note
In some cases, you may wish to reuse the command template user interface as a custom item editor.[11]

The following code sample for a command template presents a user interface allowing the user to insert a folder. This example is intentionally oversimplified to present the minimal components required to implement a command template that displays a user interface.

```
        namespace Namespace.Shell.Framework.Commands
        {
          public class NewFolder : Sitecore.Shell.Framework.Commands.Command
          {
            public override void Execute(Sitecore.Shell.Framework.Commands.CommandContext
              context)
            {
              if (context.Items.Length == 1)
              {
                Sitecore.Data.Items.Item item = context.Items[0];
                System.Collections.Specialized.NameValueCollection parameters =
                  new System.Collections.Specialized.NameValueCollection();
                parameters["id"] = item.ID.ToString();
                parameters["database"] = item.Database.Name;
                Sitecore.Context.ClientPage.Start(this, "Run", parameters);
              }
            }

            protected void Run(Sitecore.Web.UI.Sheer.ClientPipelineArgs args)
            {
              if (args.IsPostBack)
              {
                if ((!String.IsNullOrEmpty(args.Result)) && args.Result != "undefined")
                {
                  Sitecore.Context.ClientPage.SendMessage(this,"item:load(id="+args.Result
+")");
                }
              }
              else
              {
                Sitecore.Text.UrlString url = new Sitecore.Text.UrlString(
                  "/sitecore modules/shell/Namespace/NewFolder.aspx");
                url.Append("id", args.Parameters["id"]);
                url.Append("database", args.Parameters["database"]);

Sitecore.Context.ClientPage.ClientResponse.ShowModalDialog(url.ToString(),true);
                args.WaitForPostBack(true);
              }
            }
          }
        }
```

The only parameter passed to the `Execute()` method is a `Sitecore.Shell.Framework.Commands.CommandContext` object. The first element in the `Items` collection of this object identifies the `Sitecore.Data.Items.Item` that was selected when the user invoked the command template. The Execute method invokes prepares information about the processing environment, and then invokes the `Run()` method, which contains the command template logic.

The `Run()` method invokes the command template user interface dialog, and when the user completes that dialog, loads the created item into the editing user interface. In the `Run()` method, if

[11] For more information about custom item editors, see the Client Configuration Reference at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

`args.IsPostBack` is true, then the user has completed the interface, and if `args.Result` contains a value, it is the ID of the item created, and the `Run()` method causes the user interface to load that item. If `args.IsPostBack` is false, the `Run()` method invokes the command template user interface.

The following code represents the content of the user interface component file `/sitecore modules/shell/Namespace/NewFolder.aspx` referenced by the `Run()` method in the code provided previously:

```
        <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="NewFolder.aspx.cs"
Inherits="Namespace.Web.UI.NewFolder" %>
        <html>
          <head>
            <script language="javascript" type="text/javascript">
        function onClose()
        {
          window.returnValue = form1.hidCreatedId.value;
          window.close();
        }

        function onCancel()
        {
          if(confirm( "Are you sure you want to cancel?"))
          {
            window.close();
          }

          window.event.cancelBubble = true;
          return false;
        }
            </script>
            <base target="_self" />
          </head>
          <body>
            <form id="form1" runat="server">
              <div>
                <asp:textbox ID="txtName" runat="server" /><br />
                <asp:checkbox id="chkCopyInsertOptions" checked="true" runat="server"
                  Text="Copy Insert Options" />
                <asp:checkbox id="chkCopyInsertRules" checked="true" runat="server"
                  Text="Copy Insert Rules" /><br />
                <asp:button runat="server" id="btnCreate" text="Create" />
                <asp:button runat="server" id="btnCancel" text="Cancel"
                  onclientclick="onCancel();" />
                <asp:button runat="server" id="btnDone" text="Done" onclientclick="onClose();"
                  visible="false"/><br />
                <input type="hidden" runat="server" id="hidCreatedId" /><br />
              </div>
            </form>
          </body>
        </html>
```

Note
For clarity, this code uses an ASP.NET page. For greater UI consistency and developer productivity, you can implement command templates using Sitecore Sheer UI pages. The implementation of Sitecore Sheer UI is outside the scope of this document.

Note the `<base>` element and its attributes, which prevent the browser from opening a new window when the user clicks a button in the page.[12]

The following code represents the content of the code-behind for `/sitecore modules/shell/Namespace/NewFolder.aspx`:

```
        namespace Namespace.Web.UI
        {
          public partial class NewFolder : System.Web.UI.Page
```

[12] For more information about the use of HTML `<base>` element in modal windows, see http://msdn.microsoft.com/en-us/library/ms536759(VS.85).aspx.

```
        {
          protected void Page_Load(object sender, EventArgs e)
          {
            string dbName = Sitecore.Web.WebUtil.GetQueryString("database");
            Sitecore.Data.Database db = Sitecore.Configuration.Factory.GetDatabase(dbName);
            string parentID = Sitecore.Web.WebUtil.GetQueryString("id");
            Sitecore.Data.Items.Item parent = db.GetItem(parentID);

            if (String.IsNullOrEmpty(hidCreatedId.Value)
              && (!String.IsNullOrEmpty(txtName.Text) && txtName.Text != "undefined"))
            {
              chkCopyInsertOptions.Enabled = false;
              chkCopyInsertRules.Enabled = false;
              Sitecore.Data.Items.TemplateItem template =
                Sitecore.Context.ContentDatabase.Templates[Sitecore.TemplateIDs.Folder];
              Sitecore.Data.Items.Item item = parent.Add(txtName.Text, template);
              txtName.Enabled = false;
              btnDone.Visible = true;
              btnCreate.Visible = false;
              btnCancel.Visible = false;
              hidCreatedId.Value = item.ID.ToString();

              if(chkCopyInsertOptions.Checked||chkCopyInsertRules.Checked)
              {
                item.Editing.BeginEdit();

                if (chkCopyInsertOptions.Checked)
                {
                  item.Fields[Sitecore.FieldIDs.Branches].Value =
                    parent[Sitecore.FieldIDs.Branches];
                }

                if (chkCopyInsertRules.Checked)
                {
                  item.Fields["__insert rules"].Value = parent["__insert rules"];
                }

                item.Editing.EndEdit();
              }
            }
            else
            {
              if(String.IsNullOrEmpty(parent[Sitecore.FieldIDs.Branches]))
              {
                chkCopyInsertOptions.Enabled = false;
                chkCopyInsertOptions.Checked = false;
              }

              if(String.IsNullOrEmpty(parent["__insert rules"]))
              {
                chkCopyInsertRules.Enabled = false;
                chkCopyInsertRules.Checked = false;
              }
            }
          }
        }
```

Chapter 5

Insert Options

This chapter provides procedures for configuring insert options. Effective insert options control the types of items users can insert under existing items. Sitecore determines effective insert options for an item from the assigned insert options. You can programmatically manipulate effective insert options for an individual item or for all items based on a data template by assigning insert rules to the item or the standard values of the data template. You can manipulate effective insert options for all items by removing, replacing, or adding processors in the insert options pipeline.

This chapter contains the following sections:

- Assigned Insert Options vs. Effective Insert Options
- How to Assign Insert Options to a Data Template or an Individual Item
- How to Copy Insert Options from One Item to Another
- Insert Rules
- The Insert Options Pipeline

5.1 Assigned Insert Options vs. Effective Insert Options

When a user activates a feature that allows them to insert new items beneath an existing item, Sitecore invokes the insert options pipeline to determine effective insert option for the user. Effective insert options include the list of data templates, branch templates, and command templates that the user can use to insert new items under the selected item. If you do not implement any insert rules or insert options pipeline processors, then the effective insert options for a user are the assigned insert options that the user has access rights to use. Insert rules and insert options pipeline processors can dynamically manipulate the list of effective insert options.

Warning
Users with appropriate access rights can move items, duplicate items, insert items from any data template, branch template, or otherwise create items that insert options would not permit. To truly enforce an information architecture, you must restrict access rights to CMS features, or implement logic to prevent specific operations inconsistent with insert options.

5.2 How to Assign Insert Options to a Data Template or an Individual Item

To assign insert options to a data template or an individual item:

1. In the Template Manager or the Content Editor, select the standard values item or individual item.
1. Click the Configure tab.
2. In the Insert Options group, click the Assign command. The Insert Options dialog appears.
3. In the Insert Options dialog, under Options, click Templates, and then select the data templates, branch templates, and command templates to assign.
4. In the Insert Options dialog, under Options, click Insert Rules, and then select insert rules.
5. Click OK.

Important
To minimize data duplication and long-term administration, when possible, assign insert options in standard values instead of individual items.

5.3 How to Copy Insert Options from One Item to Another

To copy insert options from a source item to a target item:

1. In the Content Editor, select the source item.
2. Show the standard template fields and field raw values.[13]
3. Copy the values from the Insert Options and Insert Rules fields in the Insert Options section of the source item to the corresponding fields of the target item.
4. Hide the standard template fields and raw values.

[13] For instructions to show or hide field raw values, see the Client Configuration Cookbook at http://sdn.sitecore.net/Reference/Sitecore%206.aspx.

5.4 Insert Rules

Insert rules can dynamically alter the list of effective insert options for an item.

5.4.1 How to Create an Insert Rule

To create an insert rule:

1. Create a class that inherits from the `Sitecore.Data.Masters.InsertRule` with a constructor that accepts a single `System.Int32` parameter, and override the `Expand()` method. You can use the following code sample:

```
namespace Namespace.Data.InsertRules
{
  public class ClassName : Sitecore.Data.Masters.InsertRule
  {
    public ClassName(System.Int32 count)
    {
    }

    public override void Expand(
      System.Collections.Generic.List<Sitecore.Data.Items.Item> insertOptions,
      Sitecore.Data.Items.Item item)
    {
      //TODO: manipulate insertOptions
    }
  }
}
```

2. In the Content Editor, insert any required project-specific folders under `/Sitecore/System/Settings/Insert Rules` using the `/Templates/Common/Folder` data template.
3. In the appropriate project-specific folder, insert an insert rule definition item using the `/System/Branches/Insert Rule` data template.
4. In the insert rule definition item, in the Data section, in the Type field, enter the signature of the class.
5. To assign the insert rule to other items, see the section How to Assign Insert Options to a Data Template or an Individual Item.

5.4.2 How to Assign an Insert Rule

To assign an insert rule to all items based on a data template or to an individual item:

1. In the Template Manager or the Content Editor, select the standard values item or individual item.
2. Click the Configure tab.
3. In the Insert Options group, click the Assign command. The Insert Options dialog appears.
4. In the Insert Options dialog, under Options, click Insert Rules, and then select the insert rule.

5.4.3 Insert Rule Example: Prevent Multiple Children Based on a Single Data Template

Consider the following requirements:

- Under the home item, users can create child items based on each of a number of different data templates.
- Under the home item, users cannot create two child items based on any data template.

The following insert rule code meets these requirements:

```
        namespace Namespace.Data.InsertRules
        {
          public class PreventChildrenWithCommonTemplate : Sitecore.Data.Masters.InsertRule
          {
            public PreventChildrenWithCommonTemplate(System.Int32 count)
            {
            }

            public override void Expand(
              System.Collections.Generic.List<Sitecore.Data.Items.Item> insertOptions,
              Sitecore.Data.Items.Item item)
            {
              foreach (Sitecore.Data.Items.Item insertOption in insertOptions.ToArray())
              {
                if(item.Axes.SelectSingleItem("./*[@@templateid='"+insertOption.ID +
"']")!=null)
                {
                  insertOptions.Remove(insertOption);
                }
              }
            }
          }
        }
```

5.5 The Insert Options Pipeline

The insert options pipeline contains processors to define effective insert options. The `/configuration/sitecore/pipelines/uiGetMasters` element in `web.config` defines the insert options pipeline. At the time of this writing, the default insert options pipeline consists of three processors, each of which manipulates the list of effective insert options. These processors include:

- **`Sitecore.Pipelines.GetMasters.GetItemMasters`**: Populates the list with insert options assigned to the item or its data template's standard values.
- **`Sitecore.Pipelines.GetMasters.GetInsertRules`**: Invokes insert rules defined in insert options assigned to the item or its data template's standard values.
- **`Sitecore.Pipelines.GetMasters.CheckSecurity`**: Removes insert options that the context user does not have access rights to use.

5.5.1 Insert Options Pipeline Processors

You can manipulate effective insert options for all items using processors in the insert options pipeline.

How to Implement an Insert Options Pipeline Processor

To implement an insert options pipeline processor:

1. Create a class that implements the `Process()` method accepting an argument of type `Sitecore.Pipelines.GetMasters.GetMastersArgs`. You can use the following code sample:

```
namespace Namespace.Pipelines.GetMasters
{
  public class ClassName
  {
    public void Process(Sitecore.Pipelines.GetMasters.GetMastersArgs args)
    {
      //TODO: manipulate args.Masters
    }
  }
}
```

2. Add the processor at the appropriate point in the insert options pipeline definition element at `/configuration/sitecore/pipelines/uiGetMasters` in `web.config`.

Important
The location of each pipeline processor in the insert options pipeline affects its function. For example, the results of a processor configured before the default `Sitecore.Pipelines.GetMasters.GetInsertRules` processor are subject to insert rules, while the results of a processor configured after `Sitecore.Pipelines.GetMasters.GetInsertRules` are not. To remove insert options that the context user does not have access rights to use, the default `CheckSecurity` processor is always the last processor in the pipeline.

Insert Options Pipeline Processor Example: Allow Folders Anywhere

Consider the requirement that a user can create folders under any item.

The following insert options pipeline processor code meets this requirement:

```
namespace Namespace.Pipelines.GetMasters
{
  public class AllowFolderAnywhere
  {
    public void Process(Sitecore.Pipelines.GetMasters.GetMastersArgs args)
```

```
            {
              Sitecore.Data.Items.Item folder =
args.Item.Database.Items[Sitecore.TemplateIDs.Folder];

              if(!args.Masters.Contains(folder))
              {
                args.Masters.Add(folder);
              }
            }
          }
        }
```

Add the processor to the insert options pipeline definition after the existing `GetItemMasters` processor and before the existing `GetInsertRules` processor:

```
        <uiGetMasters argsType="Sitecore.Pipelines.GetMasters.GetMastersArgs">
         <processor mode="on"
type="Sitecore.Pipelines.GetMasters.GetItemMasters,Sitecore.Kernel"/>
          <processor mode="on"
type="Namespace.Pipelines.GetMasters.AllowFolderAnywhere,Assembly" />
          <processor mode="on" type="Sitecore.Pipelines.GetMasters.GetInsertRules,
Sitecore.Kernel"/>
          <processor mode="on" type="Sitecore.Pipelines.GetMasters.CheckSecurity,
Sitecore.Kernel"/>
        </uiGetMasters>
```

Note
You can update this example to incorporate the logic to copy insert options from the parent as described in the section Command Template Example: Copy Insert Options from the Parent Item.

Insert Options Pipeline Processor Example: Limit Children

Consider the following requirement: users cannot create more than a specified number of child items under any item.

The following insert options pipeline processor code meets this requirement:

```
        namespace Namespace.Pipelines.GetMasters
        {
          public class LimitChildren
          {
            private int _maxChildren = 25;

            public int MaxChildren
            {
              set
              {
                 maxChildren = value;
              }
              get
              {
                return _maxChildren;
              }
            }

            public void Process(Sitecore.Pipelines.GetMasters.GetMastersArgs args)
            {
              if(MaxChildren>-1 && args.Item.Children.Count>=MaxChildren)
              {
                args.Masters.Clear();
              }
            }
          }
        }
```

Add the processor to the insert options pipeline definition in `web.config` after the existing `Namespace.Pipelines.GetMasters.GetInsertRules` processor but before the existing `Namespace.Pipelines.GetMasters.CheckSecurity` processor:

```
        <uiGetMasters argsType="Sitecore.Pipelines.GetMasters.GetMastersArgs">
          <processor mode="on"
type="Sitecore.Pipelines.GetMasters.GetItemMasters,Sitecore.Kernel"/>
```

```
          <processor mode="on" type="Sitecore.Pipelines.GetMasters.GetInsertRules,
Sitecore.Kernel"/>
          <processor mode="on" type="Namespace.Pipelines.GetMasters.LimitChildren,Assembly">
            <maxChildren>10</maxChildren>
          </processor>
          <processor mode="on" type="Sitecore.Pipelines.GetMasters.CheckSecurity,
Sitecore.Kernel"/>
        </uiGetMasters>
```

Chapter 6

Aliases and Proxies

This chapter contains procedures for configuring aliases and proxies. Aliases allow multiple URLs for a content item. Proxies cause a common item or branch to appear in multiple locations.

This chapter contains the following sections:

- Aliases
- Proxies

6.1 Aliases

Aliases allow any number of URLs for a single content item. For example, by default the URL /hr/jobs.aspx activates the content item `/Sitecore/Content/Home/HR/Jobs`. You can create an alias so that in addition to the default URL /hr/jobs.aspx, the URL /jobs.aspx activates `/Sitecore/Content/Home/HR/Jobs`.

6.1.1 How to Create an Alias

To create an alias:

1. In the Content Editor, select the target item, for example `/Sitecore/Content/Home/HR/Jobs`.
2. Click the Presentation tab.
3. In the URL group, click the Aliases command.
4. Enter the alias name, for example `/jobs` or `jobs` (both are equivalent).
5. Click Add.

Note
If you use IIS7 with an application pool configured for the integrated managed pipeline mode, you can eliminate the `.aspx` extension by setting the value of the `addAspxExtension` attribute in the `/configuration/sitecore/linkManager` element in `web.config`. You do not need to use aliases to implement this change.

Warning
Using aliases results in multiple URLs for a single content item, which can confuse search engines. Use aliases consistently where they exist, or implement an alternate solution to address this challenge.

6.2 Proxies

This section provides procedures for working with proxies. A proxy cause an item or branch stored in a source location of the content tree to appear duplicated in a target location. Proxy definition items configure proxies. You can proxy a single source item in multiple locations, and you can include its descendant items in each proxied location.

Warning
Enabling proxy items can cause serious performance issues for your solution. Before you implement proxy items in a production environment, test the performance of publishing and rendering on your solution.

6.2.1 How to Enable Proxies

You must enable proxies in each database in which you use them. To enable proxies for a database:

1. Locate the `/configuration/sitecore/databases` element in `web.config` with `name` attribute corresponding to the database name.
2. Set the value of the `<proxiesEnabled>` element to `true`:

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

6.2.2 How to Create a Proxy

To create a proxy:

1. In the Content Editor, navigate to `/Sitecore/System/Proxies`.
2. Insert a proxy definition item using the `/System/Proxy` data template.
3. In the proxy definition item, in the Data section, for the Source item field, select the source item to proxy.
4. For the Target item field, select the item that will act as the parent of the proxied item.
5. For the Proxy type field, select Root item only to proxy an individual item, or Entire sub-tree to proxy an entire hierarchy of items.
6. For the Source database field, enter no value to proxy items from the same database, or a database name to proxy items from a different database.