Oracle® WebCenter Content

Developer's Guide for Site Studio for External Applications

11*g* Release 1 (11.1.1)

E13650-03

November 2011

ORACLE®

Oracle WebCenter Content Developer's Guide for Site Studio for External Applications, 11*g* Release 1 (11.1.1)

E13650-03

Primary Author: Will Harris

Contributors: Rex Young, David Truckenmiller, Ron van de Crommert

Contents

Index

Preface

The Developer's Guide for Site Studio for External Applications contains information to assist individuals who are responsible for the design and development of web sites using Site Studio for External Applications and managed by Oracle WebCenter Content Server.

Audience

This document is intended for those people identified in the organization who are responsible for designing an organization's web site managed by Site Studio.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit
http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For more information on Oracle Site Studio for External Applications and associated technologies, see the following documents:

- *Oracle WebCenter Content Site Studio for External Applications Java API Reference*
- *Oracle WebCenter Content Remote Intradoc Client (RIDC) Java API Reference*
- *Oracle WebCenter Content Services Reference Guide*
- *Oracle WebCenter Content Installation Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

About Site Studio for External Applications

Site Studio for External Applications can dramatically improve productivity and allows organizations to maintain accurate, timely, and current web content with consistent branding and presentation across all corporate sites. With Site Studio, you can centralize control of site architecture and presentation, while distributing content development and ongoing maintenance to business units or other teams. Additionally, Site Studio for External Applications can be integrated with third-party applications servers.

This section covers the following topics:

1.1 New for this Release

These enhancements, upgrades, or changes are new for this release:

- Site Studio projects can be bundled and deployed from JDeveloper to a remote WebSphere application server. See Section 5, "Deploying Your Site Studio Project to a Remote Application Server" for more information.
- Site Studio for External Applications can be used on Oracle Content Server 10*g*R3. See Section 1.3, "Installing Site Studio for External Applications" for more information.
- JSPX templates can be stored on Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3. See Section 1.10, "Understanding Site Studio JSP/JSPX Templates" for more information.

- Oracle Site Studio Manager functionality can be added to your site. See Section 1.11, "Including Oracle Site Studio Manager Functionality" for more information.
- The Fusion Middleware Configuration Wizard provides an option to generate a domain configured automatically to support Oracle WebCenter Content - SSXA Server. For additional information, refer to the *Oracle WebCenter Content Installation Guide* or the Fusion Middleware Configuration Wizard online help.
- Preference settings include the automatic check in of template files and definition files on editor save of those files. These preference settings are enabled by default and can be changed on the preferences panel (from the JDeveloper main menu, select **Tools**, then **Preferences**, and then **SiteStudio**).

1.2 Introduction to Site Studio for External Applications

Site Studio for External Applications is a powerful, flexible web development tool that offers a comprehensive approach to designing, building, and maintaining enterprise-scale web sites in an application server environment. Web site development and template creation is performed in the JDeveloper environment. Site assets, such as files and graphics, are stored and managed in the content server. Templates may be stored and managed in the content server or as part of your project in JDeveloper, depending on the type of template used.

Site Studio JSP/JSPX templates, including page templates, region templates, and subtemplates, can be stored locally within Oracle JDeveloper or checked in to your Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3 instance.

- JSPX templates stored locally are typically located in a subdirectory within your project on JDeveloper. For example, the `/wcm/templates/page/` subdirectory would typically contain all page templates associated with your project.
- JSPX templates can also be stored on an Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3 instance. This allows templates to be modified and updated on your site without having to re-deploy the application. JSPX templates stored on Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3 can be edited in your JDeveloper environment and previewed multiple times before being committed to Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3.

Your project may contain both JSPX templates stored locally and JSPX templates stored on Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3. However, it is recommended that you build each project using one template type.

1.3 Installing Site Studio for External Applications

Site Studio for External Applications requires two extensions be installed on Oracle JDeveloper and two components be enabled on Oracle WebCenter Content Server.

Follow the steps provided in these sections to install the required extensions and components:

- Section 1.3.1, "Installing Required Extensions on Oracle JDeveloper"
- Section 1.3.2, "Enabling Required Components on Oracle WebCenter Content Server 11gR1"
- Section 1.3.3, "Enabling Required Components on Oracle Content Server 10gR3"

1.3.1 Installing Required Extensions on Oracle JDeveloper

Site Studio for External Applications 11*g*R1 requires that the Remote Intradoc Client (RIDC) extension and Site Studio for External Applications (SSXA) extension for Oracle JDeveloper be installed.

Required Extensions

These extensions must be installed on Oracle JDeveloper:

- Oracle Remote Intradoc Client (RIDC) extension 11.1.1.6 (11*g*R1 PS5). The Remote Intradoc Client (RIDC) extension for JDeveloper is included with the Remote Intradoc Client (RIDC) suite distribution.

  The RIDC extension (oracle.ucm.ridc.jdev-11.1.1.6.zip) is located in the modules/jdev directory of the RIDC suite distribution (ridc-suite-11.1.1.6.zip).

- Oracle Site Studio for External Applications (SSXA) extension 11.1.1.6 (11*g*R1 PS5). The Site Studio for External Applications (SSXA) extension for JDeveloper is included with the Web Content Management (WCM) suite distribution.

  The SSXA extension (oracle.ucm.wcm.jdev-11.1.1.6.zip) is located in the modules/jdev directory of the Web Content Management (WCM) suite distribution (wcm-suite-11.1.1.6.zip).

The Remote Intradoc Client (RIDC) suite distribution and Web Content Management (WCM) suite distribution can be downloaded from the Oracle Technology Network (OTN) at http://otn.oracle.com.

Installation steps

Follow these steps to install the required extensions on Oracle JDeveloper 11*g*R1:

1. Download the Remote Intradoc Client (RIDC) suite distribution and Web Content Management (WCM) suite distribution from Oracle Technology Network (OTN).
2. Unbundle the Remote Intradoc Client (RIDC) suite distribution (ridc-suite-11.1.1.6.zip) to a location on the system hosting your Oracle JDeveloper instance.
3. Unbundle the Web Content Management (WCM) suite distribution (wcm-suite-11.1.1.6.zip) to a location on the system hosting your Oracle JDeveloper instance.
4. From the JDeveloper main menu, select **Help** then **Check for Updates**.
5. Enable the **Install from Local File** option.
6. Click **Browse** and navigate to the unbundled Remote Intradoc Client (RIDC) suite distribution (ridc-suite-11.1.1.6.zip).
7. Select the RIDC extension (oracle.ucm.ridc.jdev-11.1.1.6.zip) located in the modules/jdev directory and click **Open**.
8. Click **Next** to install/update the extension.
9. Click **Finish**.
10. From the JDeveloper main menu, select **Help**, then **Check for Updates**.
11. Enable the **Install from Local File** option.
12. Click **Browse** and navigate to the unbundled Web Content Management (WCM) suite distribution (wcm-suite-11.1.1.6.zip).

13. Select the SSXA extension (oracle.ucm.wcm.jdev-11.1.1.6.zip) located in the modules/jdev directory and click **Open**.
14. Click **Next** to install/update the extension.
15. Click **Finish**.

For additional information on installing JDeveloper extensions refer to the *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper* or the Oracle JDeveloper online help.

1.3.2 Enabling Required Components on Oracle WebCenter Content Server 11*g*R1

Site Studio for External Applications 11*g*R1 requires that the SiteSudio component and SiteStudioExternalApplications component be enabled on Oracle WebCenter Content Server.

Both the SiteStudio and SiteStudioExternalApplications components are installed as part of the full Oracle UCM installation and simply need to be enabled on Oracle WebCenter Content Server 11*g*R1.

If you are planning to use dynamic conversion of native documents on your Site Studio web sites, then you must configure Dynamic Converter in Oracle WebCenter Content Server.

Required Components

These components must be enabled on Oracle WebCenter Content Server 11*g*R1:

- SiteStudio 11.1.1.6 (11*g*R1 PS5).
- SiteStudioExternalApplications 11.1.1.6 (11*g*R1 PS5).

Refer to the Oracle WebCenter Content Server documentation or online help for instructions on enabling components.

1.3.3 Enabling Required Components on Oracle Content Server 10gR3

Site Studio for External Applications can be used with Oracle Content Server 10*g*R3 platform. However, because the Universal Content Management 11*g*R1 distribution is considerably different from the 10*g*R3 distribution, several manual actions must be performed.

Required Updates and Additional Configuration

These updates and additional configurations must be completed:

- Oracle Content Server 10*g*R3 with the latest 10gR35CoreUpdate and 10gR35NativeUpdate bundles installed. These bundles update Oracle Content Server and Dynamic Converter to the required feature level and the web filter to the required version.
- SSUrlMapPlugin.dll (.so) 11.1.1.3.0 or greater

  This DLL is included with the 10gR35NativeUpdate bundle. To confirm the version number, from the Content Server Admin menu, select **Filter Administration**.
- Dynamic Converter 8.1.0.736 or greater

  If you are planning to use dynamic conversion of native documents on your Site Studio web sites, you must install and configure Dynamic Converter build version 8.1.0.736 or greater in Oracle Content Server.

- JDK version 1.6 must be installed (Oracle Content Server 10gR3 is shipped with JDK 1.5).
- The Oracle Content Server configuration file *Intradoc.cfg* must be updated to reference the new JDK 1.6 installation directory. Add the entry to the `#Additional Variables` section of the configuration file (for example, `JDK_custom="C:\Program Files\Java\SDK 1.6.0"`).
- The JDeveloper Site Studio for External Applications connections dialog is pre-populated with a default web connection port value of 16200 for an Oracle WebCenter Content Server 11*g* instance. Oracle Content Server 10*g*R3 web servers most often listen on port 80. As such, this value will likely have to be changed when connecting to an Oracle Content Server 10*g*R3 instance.

Required Components

These components must be installed and enabled on Oracle Content Server10*g*R3:

- SiteStudio 11.1.1.6 (11*g*R1 PS5).

  The SiteStudio 11.1.1.6 component is provided with the Site Studio suite distribution (sitestudio-11.1.1.6.zip).

- SiteStudioExternalApplications 11.1.1.6 (11*g*R1 PS5).

  The SiteStudioExternalApplications 11.1.1.6 component (oracle.ucm.wcm.idc-component-11.1.1.6.zip) is located in the /component directory of the Web Content Management (WCM) suite distribution (wcm-suite-11.1.1.6.zip).

The Site Studio suite distribution and Web Content Management (WCM) suite distribution can be downloaded from the Oracle Technology Network (OTN) at http://otn.oracle.com.

Refer to the Oracle Content Server 10*g*R3 documentation or Oracle Content Server online help for instructions on installing and enabling components.

1.4 Understanding the Site Studio Project Structure

This is the basic structure of a Site Studio project in the JDeveloper environment:

```
+ Site Files
     + templates
       + page
       + region
       + subtemplates
+ Web Content
     + wcm
     + WEB-INF
       + sites
       - wcm-config.xml
       - web.xml
       - weblogic.xml
```

Please note the following nodes and files:

- **Site Files**: This is a virtual node that points to the Web Content/wcm node.
- **Web Content/wcm**: This node contains the page templates, region templates, and subtemplates for this project.
- **Web Content/WEB-INF**: This node contains the XML descriptor files and configuration files.

- **WEB-INF/sites**: This node contains the XML files that store the data and configuration information for the Site Studio site. See Section 1.5, "Understanding Site Studio Site Files" for more information.
- **WEB-INF/wcm-config.xml**: The Site Studio configuration file.
- **WEB-INF/web.xml**: The web-app XML descriptor file.
- **WEB-INF/weblogic.xml**: The Oracle WebLogic deployment descriptor file.

1.5 Understanding Site Studio Site Files

In JDeveloper, Site Studio site files are located in the `sites` directory of your Site Studio project (from the main menu, select **View**, then **Application Navigator**, then **Projects**, then expand **Web Content**, then expand **WEB_INF** and then expand **sites**).

Site Studio site files are XML files that store the data and configuration information for the Site Studio site. These XML files are maintained on JDeveloper, but all content resides on the content server.

The site files contain site-related information including:

- The site hierarchy.
- Properties of each site section (including associated page templates, region templates, and content files, and custom section properties).
- Explicit data file associations (for example, which content files are used and their location in the site).
- Mappings of placeholder names to placeholder definitions.
- Items in the asset pane (including how they are categorized).

1.6 Understanding the Site URL Format

The site URLs are hierarchical, allowing navigation through a given project file. They have the following general format:

```
/[Site Identifier]/[Section 1]/[Section 2]/.../[content].html
```

- **Site Identifier**: The path that identifies a site, the URL pattern listed in the wcm-config.xml file.
- **Section**: A list of sections, identified by either the `urlDirName` attribute or the `label` attribute in the project file.
- **Content**: Can be the content ID (dDocName) of an item on Oracle WebCenter Content Server, or `index` to refer to the primary page.

For example, a path to the primary page of the About section:

```
/mysite/About/index.html
```

A path to a piece of content to display in the Products section:

```
/mysite/Products/new_product.html
```

1.7 Understanding Security and Contributor Authentication

The security check is performed in the Site Studio for External Applications layer. The web browser user (typically anonymous) is taken as the `userId` and each datafile is checked by the Site Studio for External Applications layer to evaluate whether that

userId has the equivalent permission through the content server's web interface. The adminUser's userId is used to fetch the datafile from the content server and store it in the Site Studio for External Applications cache.

Security Settings

In the wcm-config.xml file, the security element defines how security is handled. If the enabled attribute is set to false, no security checks are performed in consumption mode and all content displayed through the placeholder will be readable. Requests through the proxy servlet will still be validated.

Depending on your content server configuration, you can also specify the level of security using the securityType attribute (each level is in addition to the previous level):

- **groups**: Check security based on the content server security group.
- **accounts**: After checking the group, check the security accounts.
- **acl**: After checking the accounts, validate against any access control lists (ACLs).

Contribution and Design Mode Authentication

Contribution and design modes require authentication to the application server running the site. This is placed into the web.xml file automatically, to verify these settings review the web.xml file. See the topic *Assigning Security Roles for the Contributor and Designer* in the online help for details on contribution mode and design mode.

For example:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>ContributionMode</web-resource-name>
    <url-pattern>/wcm-contrib/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>WCMContributor</role-name>
  </auth-constraint>
</security-constraint>

<login-config>
  <auth-method>FORM</auth-method>
  <realm-name>OpenWCM</realm-name>
  <form-login-config>
    <form-login-page>/wcm/support/login/wcm-login.jsp</form-login-page>
    <form-error-page>/wcm/support/login/wcm-login.jsp</form-error-page>
  </form-login-config>
</login-config>

<security-role>
  <role-name>WCMContributor</role-name>
</security-role>
```

Upon selecting **Ctrl+Shift+F5** (the default keyboard shortcut to enter contribution mode) on a page with placeholder content, the page is refreshed to a URL with /wcm-contrib as the first segment. The application server then validates that the user has the appropriate application role assigned to them via the application server. The role name, as described in the web.xml, is WCMContributor. If the user does not have this role, a login screen is presented and the application server denies access until the credentials are validated and the role condition is met.

The default login page can be customized by changing the web.xml to point at a customized JAAS login page or by adding a local copy of the file `/wcm/support/login/wcm-login.jsp` to your web application. This page will then be used instead of the shared default version.

Once in contribution mode or design mode, the user ID is now used to validate security and all placeholder content is retrieved live from the content server. Additionally, the rendition is switched from *Latest Released* to *Latest* so the contributor user can view content in workflow or otherwise not yet released.

1.8 Understanding Content Caching

Content is cached in a temporary location on the application server. This can be customized to a particular location using the `stagingDir` attribute on the `staging` element in the wcm-config.xml file. The content cache stores a local copy of all files, associated metadata, and any associated conversions or renditions.

Any request for content first checks the local cache. If not found, the content server is contacted and the content is placed into the cache. Note that no security information is cached; all security calls go to the content server. Additionally, only *Latest Released* content is cached; content retrieved during contributor or design modes (*Latest Rendition*) is not put into the cache.

A background thread, controlled by the `contentServer` element using the `pollerInterval` attribute, pings the content server at regular intervals to keep the cache content up to date. A content server administrator can request all caches be refreshed by visiting the Cache Administration page on the content server at this URL (example):

```
http://localhost:16200/cs/idcplg?IdcService=SSXA_GET_ADMIN_PAGE
```

1.9 Understanding Service Caching with the Site Studio Tag Library

The caching facilities in Site Studio allow for the caching of service calls. Any service call can be cached to the same location that content and metadata is currently cached. The caching facility is defined using DataBinder properties:

1. Specify a special parameter to enable caching.
2. Determine the cache key using one of the following methods:
   - Specify a custom cache key.
   - Specify the local data fields to use to generate a key.
   - Allow the system to calculate the cache key by using a hash of all local data.

Caching Properties

All caching is controlled using properties set in the DataBinder. For JSP/JSPX, these properties can be added with the `wcm:idcParameter` tag to either a `wcm:idcService` call or a `wcm:dynamicList` call.

Parameters:

- `__ssxaCacheEnabled`: Set to `true` to enable caching.
- `__ssxaCacheKey`: Specify a cache key to use for this request. Optional.
- `__ssxaCacheFields`: Specify a comma-separated values (CSV) list of fields to use to generate the cache key (if not specified, all local data is used). Optional.

- `__ssxaCacheTTL`: Specify a time to live value, in milliseconds, for the cached data. If not specified, the default value of 5 minutes is used. If value of 0 (zero) or less is specified, the cache item will live indefinitely (no timeout). Optional.

Example of caching a dynamic list, with the value being refreshed every 60 seconds:

```
<wcm:dynamicList var="searchResponse" element="nativedocs">
  <wcm:idcParameter name="__ssxaCacheEnabled" value="true" />
  <wcm:idcParameter name="__ssxaCacheTTL" value="60000" />
</wcm:dynamicList>
```

Example of caching a `GET_SEARCH_RESULTS` service call:

```
<wcm:idcService var="dataBinder" service="GET_SEARCH_RESULTS">
  <wcm:idcParameter name="QueryText" value="" />
  <wcm:idcParameter name="__ssxaCacheEnabled" value="true" />
</wcm:dynamicList>
```

Cache Details

Service calls are staged into the cache directory, alongside the content, metadata and project files. The directory `/idc-service` is created with this directory structure:

```
/idc-service/service_name/user_name/cacheID.hda
```

This example shows a cache directory for a `GET_SEARCH_RESULTS` service call:

```
+- idc-service/
   +- ssxa_get_dynamic_list_results/
      +- anonymous/
         +- 3164744485.hda
   +- get_search_results/
      +- anonymous/
         +- 2452981209.hda
```

1.10 Understanding Site Studio JSP/JSPX Templates

Site Studio JSP/JSPX templates, including page templates, region templates, and subtemplates can be stored locally within JDeveloper or remotely on Oracle WebCenter Content Server.

- JSPX templates stored locally are typically located in a subdirectory within your project on JDeveloper. For example, the `/wcm/templates/page/` subdirectory would typically contain all page templates associated with your project. Templates to be stored locally are created from the JDeveloper New Gallery dialog. See Section 1.10.1, "Creating a Local JSPX Template" for more information.
- JSPX templates can also be stored on an Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3 instance. This allows templates to be modified and updated on your site without having to re-deploy the application. JSPX templates stored on Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3 can be edited in your JDeveloper environment and previewed multiple times before being committed to Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3. Templates to be stored remotely on Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3 are created from the Site Studio Site Assets window. See Section 1.10.2, "Creating a Remote JSPX Template" for more information.

Your project may contain both JSPX templates stored locally and JSPX templates stored on Oracle WebCenter Content Server 11*g*R1 or Oracle Content Server 10*g*R3. However, it is recommended that you build each project using one template type.

Template Syntax and Structure

Any template type can contain any valid JSP syntax. However, it is assumed that the templates will have the following structure:

- **Page Template**: Contains `wcm:placeholder` tags that create areas in a page for assignable content.
- **Region Template**: For modeling data file content. Typically contains either a `wcm:dataFile` tag or `wcm:dynamicConversion` tag to retrieve the assigned content. It does not contain `wcm:placeholder` tags.
- **Subtemplate**: Like a page template, but is typically responsible for a portion of a page after being assigned to a placeholder. May contain other `wcm:placeholder` tags.

Template Registration

Templates are registered with a unique identifier in the wcm-config.xml file. This identifier is used to identify the template throughout the Site Studio application. The identifiers are used in the project file to identify page templates (either primary or secondary) and are used in placeholder definitions, to identify a region template or subtemplate.

This template maps the name `homepage` to the path `/wcm/templates/page/homepage.jspx`. When the homepage identifier is encountered, the `RequestDispatcher` is invoked to include the path.

```
<mappings>
  <pageTemplates>
    <mapping path="/wcm/templates/page/homepage.jspx" id="homepage"/>
  </pageTemplates>
</mappings>
```

There is no requirement that the template has to point to a JSP or JSPX file. The mapping can point to any valid web application resource. For example, a section could point to another servlet for processing:

```
<mappings>
  <pageTemplates>
    <mapping path="/servlets/reserveMeetingRoom.do" id="reserveMeeting"/>
  </pageTemplates>
</mappings>
```

1.10.1 Creating a Local JSPX Template

Follow these steps to create a JSPX template stored locally on JDeveloper (typically, in a subdirectory within your project on JDeveloper).

1. From the main menu, select **View**, then **Application Navigator**.
2. In the Application Navigator, expand the **Projects** panel.
3. Select a Site Studio project.
4. Expand **Web Content**, then **wcm**, and then **templates**.
5. Right-click page and select **New**.

   The New Gallery dialog opens.
6. From the Categories list, expand **Web Tier**, then select **JSP**.
7. From the Items list, select **JSP**.

8. Click **OK**.

   The Create JSP dialog opens.

9. For File Name, provide a descriptive file name that identifies this file as a template for this site.

10. For Directory, accept the default or provide a different directory. By default, the Site Studio template directory in JDeveloper is used. If you want to store the files outside of the JDeveloper directory structure, specify the location.

11. Select **Create as XML Document**. This creates a JSP page written in XML syntax using JSPX as the file extension.

12. Select **Register Site File** to register the template with your site. In most cases, you will want your template registered with your site.

13. Select the template type from the Asset Type drop-down list.

14. Provide a site file ID that identifies this file as a template for this site.

15. Provide a description for the new template.

16. Click **OK**.

17. From the main menu, select **File**, then select **Save All**.

See the topic *Creating and Editing Local Templates* in the online help for additional information on creating and editing JSPX templates stored locally on JDeveloper.

1.10.2 Creating a Remote JSPX Template

Follow these steps to create a remote JSPX template stored on Oracle WebCenter Content Server:

1. From the main menu, select **View**, then **Site Studio**, and then **Site Assets**.

2. From the Project drop-down list (top left), select your project.

   If your project is not listed, it means you are not logged in to your connection. From the Application Resources panel, expand **Connections**, then **Site Studio**, then right-click the connection and select **Login**.

3. From the Asset Type drop-down list (top right), select a template type. For example, Page Template, Region Template, or Subtemplate.

4. Click the **Create New Asset Type** button and select **New JSPX Template**.

5. On the Create New Asset dialog, enter a title and content ID. For example, `ss-homepage-pt`.

6. Click **OK**.

7. On the Site Asset window, click the **Refresh** button to view your new Content Server template.

See the topic *Creating and Editing Remote Templates* in the online help for additional information on creating and editing remote JSPX templates stored on Oracle WebCenter Content Server.

1.11 Including Oracle Site Studio Manager Functionality

Oracle Site Studio Manager provides an environment in which site managers can make changes to the site hierarchy such as adding and removing sections, changing the page template assigned to a section, and modifying several site properties.

Oracle Site Studio Manager functionality can be enabled on external sites that use Site Studio for External Applications. However, due to the different environment, Manager functionality in an external application has several differences from Oracle Site Studio Manager running in an Oracle WebCenter Content Server instance. These differences include how Manager functionality is provided and launched, how externally managed JSP/JSPX page templates are handled, and how preview URLs are generated.

Including Oracle Site Studio Manager in Your Site

To support Oracle Site Studio Manager functionality with Site Studio for External Applications, a *site-manager* page is provided. This page hosts the Oracle Site Studio Manager functionality. To include Oracle Site Studio Manager functionality in your site, you must provide a link from your site to the `site-manager.jsp` file.

Note: The code example used in this section assumes you have created a section on your site called *Manager* and that this section is marked as a *contributor-only section* so that it is only visible in contribution mode.

To do this, the page template assigned to the Manager section must include the following code:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
  xmlns:c="http://java.sun.com/jsp/jstl/core"
  xmlns:wcm="http://www.oracle.com/jsp/wcm">
<c:redirect url="/wcm-contrib/wcm/support/manager/
  site-manager.jsp?siteID=${wcmContext.siteID}" />
</jsp:root>
```

The `redirect` tag redirects the browser to the `site-manager.jsp` file. When a user navigates to the Manager section of the site they will automatically be redirected to this page.

Refer to the *Administrator and Manager's Guide for Oracle Site Studio* for details on using Oracle Site Studio Manager and additional information on Oracle Site Studio Manager functionality.

1.12 Enabling HCSP-Based Custom Element Forms

Mime-mapping changes are required when using Site Studio for External Application with HCSP-based custom elements.

Within Site Studio for External Applications, custom element forms are HTML files that define custom forms for use in elements (for example, selection forms for specific file types). Hyper Content Server Page (HCSP) forms contain HTML and Idoc Script code (a proprietary scripting language) and are often used to request services from the content server. HCSP-based custom element forms use the .hcsp file extension. By default, HCSP-based custom element forms will not load in the Site Studio for External Applications JDeveloper environment and the user is prompted to download and save the file. If you want to use HCSP-based custom element forms, you need to add a mime-mapping to the Site Studio project web.xml file (not the UCM web.xml file) to enable the custom element forms to be executed instead of being downloaded.

Follow these steps to add a mime-mapping to the Site Studio project web.xml file.

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project.
3. Expand **Web Content**, and then expand **WEB-INF**.
4. Right-click **web.xml** and select **Open**.
5. Select the **Source** tab.
6. Add the following mime-mapping at the end of the web.xml file before the closing web-app tag:

```
<mime-mapping>
  <extension>hcsp</extension>
  <mime-type>text/html</mime-type>
</mime-mapping>
```

7. From the main menu, select **File** and then **Save All**.

2

Understanding Site Studio Web Sites

Site Studio provides a powerful set of tools that can allows you to design, build, and maintain web sites efficiently. To get the most out of these tools, it is useful to understand some basic concepts about Site Studio web sites.

This section covers the following topics:

2.1 Separation of Site Presentation and Content

One thing that makes Site Studio a valuable tool is that it allows web site content to be completely separate from web site presentation. This way, different people can manage and be responsible for the information on a web site without inadvertently affecting the layout, design, or look-and-feel of the site. In addition, the people assigned to manage the site content can make changes as necessary without having to send them to someone else to complete the task. This removes an important bottleneck that exists in many site management scenarios, where all site changes must be handled by a very limited number of site administrators.

With separation of site presentation and content in mind, the files associated with a web site can be divided into three main categories:

- Section , "Site Presentation Files"
- Section , "Site Content Files"
- Section , "Site Control and Configuration Files"

Site Presentation Files

A number of the files associated with a Site Studio web site are used to define what the site looks like in terms of page layout and formatting. They provide the design framework within which the site content is displayed. Any changes to these files typically affect the entire site (or large portions of it), and they are usually created and managed by dedicated site designers.

Site Studio uses the following files for site presentation:

- **Page templates**: Fully-formed HTML files that define the layout and high-level look-and-feel of web pages, including the placement of contribution regions (that is, editable areas on the page), navigation aids (in the form of fragments) and site-wide images (banners and the like). Page templates are the highest-level site design object. For more information, see Section 2.12, "Page Templates."
- **Region templates**: Partial HTML files (that is, without head and body sections) that define the layout and look-and-feel of the data in contribution regions within web pages. For more information, see Section 2.9, "Region Templates and Region Definitions."
- **Subtemplates**: Partial HTML files (that is, without head and body sections) that can be inserted into placeholders on page templates to divide them into further smaller, reusable areas with their own placeholders and contribution regions. For more information, see Section 2.11, "Subtemplates."

In addition to these files, which directly affect the site presentation, there are also several files configuration files that have an impact on web site presentation (see "Site Control and Configuration Files" on page 2-3).

Site Content Files

The site content (that is, the actual information on the site) is stored in separately managed files, separate from the presentation context in which they appear. This enables them to be managed separately and reused within a web site, or even between web sites (providing these sites are all managed using the same content server).

Site Studio uses the following files for site content:

- **Contributor data files**: Content files in XML format that are generated by Site Studio. Contributor data files are edited using the Site Studio Contributor application. For more information, see Section 2.13, "Contributor Data Files and Native Documents."
- **Native documents**: Content files created using familiar third-party applications such as Microsoft Word. Native documents are converted to HTML format using Dynamic Converter, and they are edited using their associated application. For more information, see Section 2.13, "Contributor Data Files and Native Documents."
- **Images**: Graphic files (JPG, GIF, PNG) that are included in content files or page templates (for example, corporate banners or product images).

- **Other media**: Any other media files that could be used on a web site, such as Flash animations, video files, audio files, and so on.

Site Control and Configuration Files

In addition to the files that directly affect site presentation, there are several configuration files that also have an impact on web site presentation.

Site Studio uses the following files for site control and configuration:

- **Element definitions**: Files that define the editing experience for element types. Specifically, they specify what a contributor can do when editing an element. For more information, see Section 2.8, "Elements and Element Definitions."
- **Region definitions**: Files that define the type of content that elements of a particular type consists of. They also specify the content creation and switching options available to contributors for contribution regions, and set default metadata for content files associated with these regions. For more information, see Section 2.9, "Region Templates and Region Definitions."
- **Placeholder definitions**: Files that define what region definitions, region templates, and subtemplates are allowed for the associated placeholders. They also specify what contributor actions are allowed for the placeholders. For more information, see Section 2.10, "Placeholders and Placeholder Definitions."
- **Validation scripts**: JavaScript files that define the validation rules for element data to determine that the data meets the requirements (for example, it does not exceed a certain maximum length or contain some illegal characters).
- **Conversion definitions**: Files that specify the conversion rules for native documents on a web site.
- **Custom configuration scripts**: JavaScript files that override the default Contributor editor configuration to provide contributors with a customized editing experience.
- **Custom element forms**: HTML files that define custom forms for use in elements (for example, selection forms for specific file types). Site Studio comes with several predefined custom element forms. (These forms are also checked into the content server when the Site Studio component is installed.)

2.2 Site Asset Storage

All files associated with a Site Studio web site are stored and managed using Oracle WebCenter Content Server. A number of custom metadata fields specific to Site Studio are used to specify where and how the files are used.

Figure 2–1 Site Studio Metadata Fields on Content Information Page of Site Asset

Site Studio uses the following metadata fields for all site-related files:

- **Web Site Object Type**: Specifies the type of file for the web site (for example, "Data File," Stylesheet," "Region Template," and "Placeholder Definition").
- **Web Sites**: Lists the web site(s) that the file is associated with. This means that the file can be used on the listed site(s), although it is not necessary. A file may be associated with multiple web sites, which means it can be reused between sites. This makes multi-site management more efficient.
- **Exclude From Lists**: Lists the web site(s) for which a contributor has specified that a particular content file (a contributor data file or native document) should not display in dynamic lists on the web site.
- **Web Site Section**: Specifies where a content file is displayed by default on a web site (unless a target section is explicitly specified in the original hyperlink).
- **Region Definition**: Specifies the region definition that a region template or content file (contributor data file or native document) is associated with. This determines how the file displays on the site and what contributors are allowed to do. A region template and content file can be associated with only one region definition, but a region definition may have many region templates and content files associated with it.

Please note that some fields are automatically set as the file is created and checked into the content server. Also, not all metadata fields are used for all site assets, since they may not be relevant. For example, the xRegionDefinition metadata field is not used for page template, since these are not associated with region definitions.

You can change the values of these fields on the content information page of a file, but be carefully when doing that, since this may affect the way the file is used on a site.

2.3 Site Roles

When the various roles for web site creation and management within the organization are determined, each can focus on specific tasks in making the web site work.

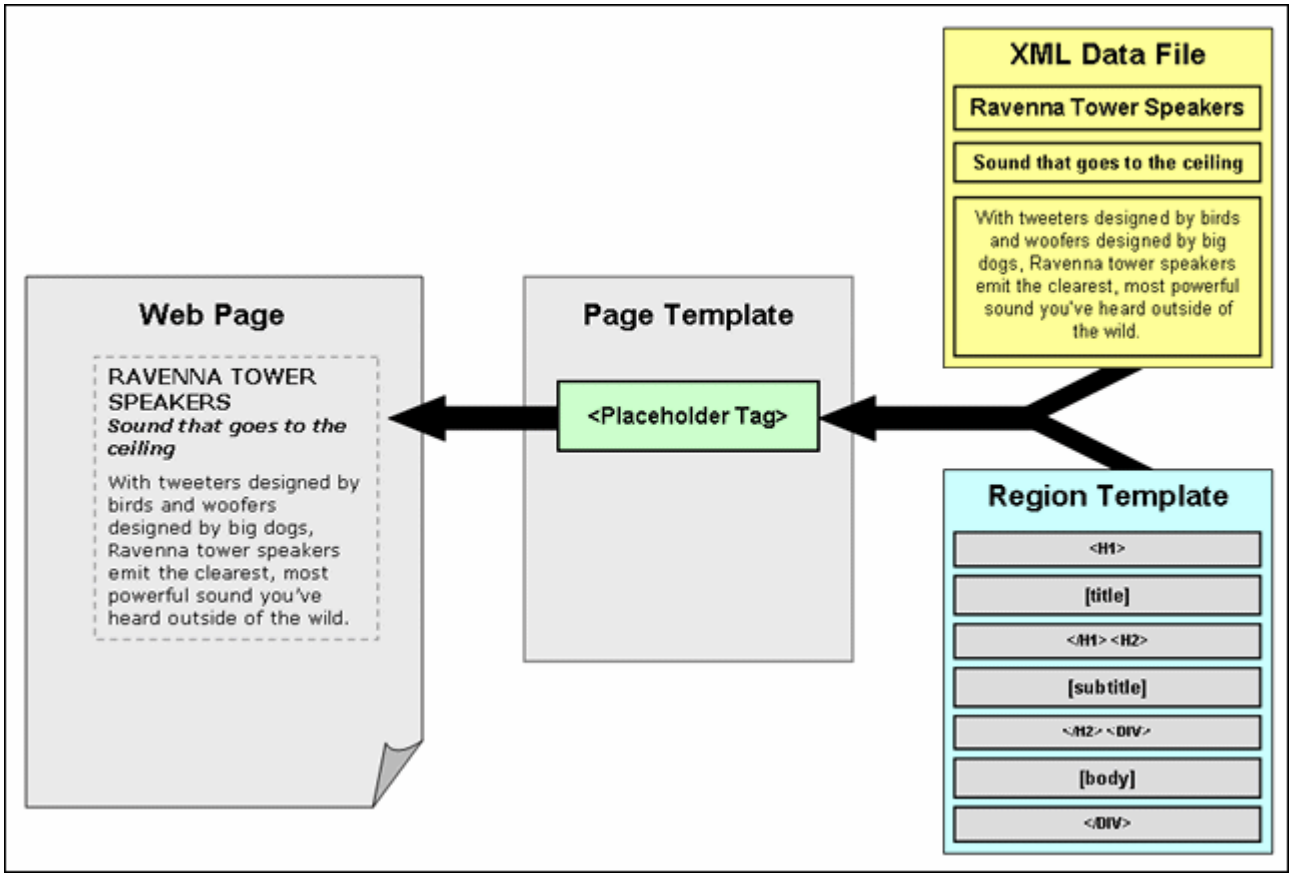
Designers focus on how the web site looks. That is, the structure of the pages, the way the pages are laid out, the design, and the corporate identity.

Contributors are then able to place the content on the page without having to code the pages. Contributors can then update and edit the content without affecting how the page is displayed, and without having to make multiple changes to multiple areas of the web site. The designer, for the most part, does not control the content. Contributors, for the most part, do not control how the content displays on the site.

2.4 Presentation Model

Site Studio completely separates the presentation layer from the content layer of a web site, as shown in Figure 2–2.

Figure 2–2 Presentation Model for Site Studio Web Sites



Page templates are used to define the site framework within which content is displayed. They contain standard HTML layout and formatting code, along with Site Studio tags to specify where fragments and/or placeholders go. Placeholders specify where the contribution regions (that is, editable areas) are on the page. Please note that they do not specify anything about what goes in these regions, both in terms of content and visual presentation. That is handled by region templates (with their associated region definitions).

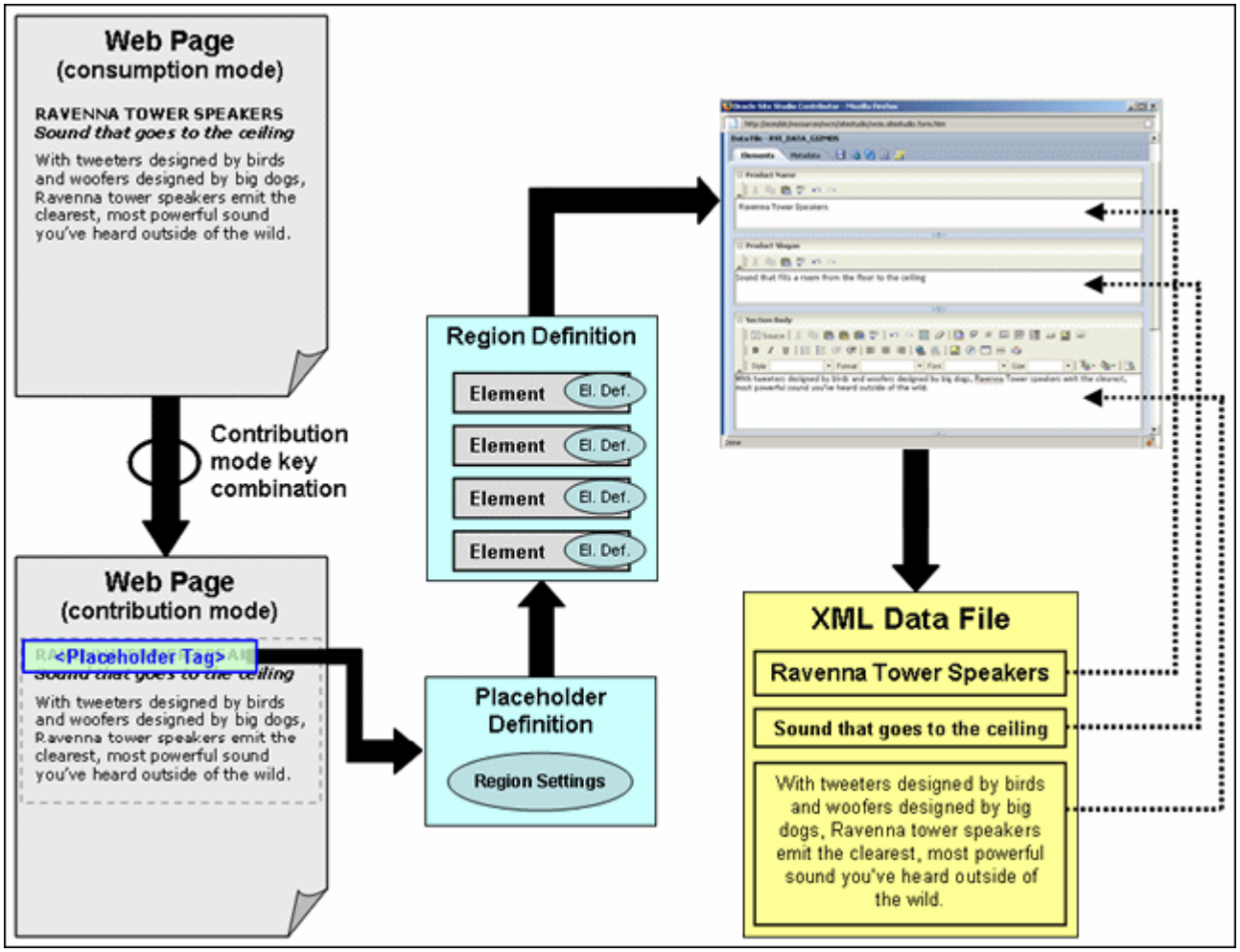
Region templates define the layout and look-and-feel of the data in a contribution region (marked on a page template using a placeholder tag). They are separately managed site assets, which makes them easy to reuse within web sites, or even between web sites. (Please note that in Site Studio releases prior to 10*g*R4 region presentation was not separately managed, but included within page templates, or 'layout pages' as they were called in these previous releases.)

The content of contribution regions is stored in data files, which are also separately managed site assets. When it is time to generate a web page, Site Studio looks at the placeholder on the page template, takes its associated region template and data file, and merges these two to create HTML code that is inserted into the page template at the position of the placeholder tag. This creates the final web page, which all content in place, presented and formatted in accordance with the site and page settings.

2.5 Contribution Model

As with the presentation model, the contribution side of Site Studio web sites separates site content from presentation, as shown in Figure 2–3.

Figure 2–3 Contribution Model for Site Studio Web Sites



When site contributors decide that they want to edit the contents of a web page, they press a key combination to reload the page in contribution mode. (The default key combination is Ctrl+Shift+F5, but this may be modified.)

After a page is in contribution mode, all contribution regions (that is, editable areas) on a web page are marked and underlying code has been added to the page to identify what placeholder is associated with the contribution region. Based on this information, Site Studio can establish what placeholder definition and region definition are used for the placeholder. The region definition identifies the structure of the content in the contribution region in terms of its constituent data segments (elements). Each element

has an element definition, which defines what editing options are available to contributors for the element.

When the contributor decides to edit the content in a contribution region, its associated contributor data file is checked out of the content server. The structure of the data file matches that of its associated region definition in terms of number and types of elements. The data is loaded from the data file and presented in the Contributor editor. Each element in the data file is presented as an editable element in the editor, with the editing features as defined in the element definition for the element type.

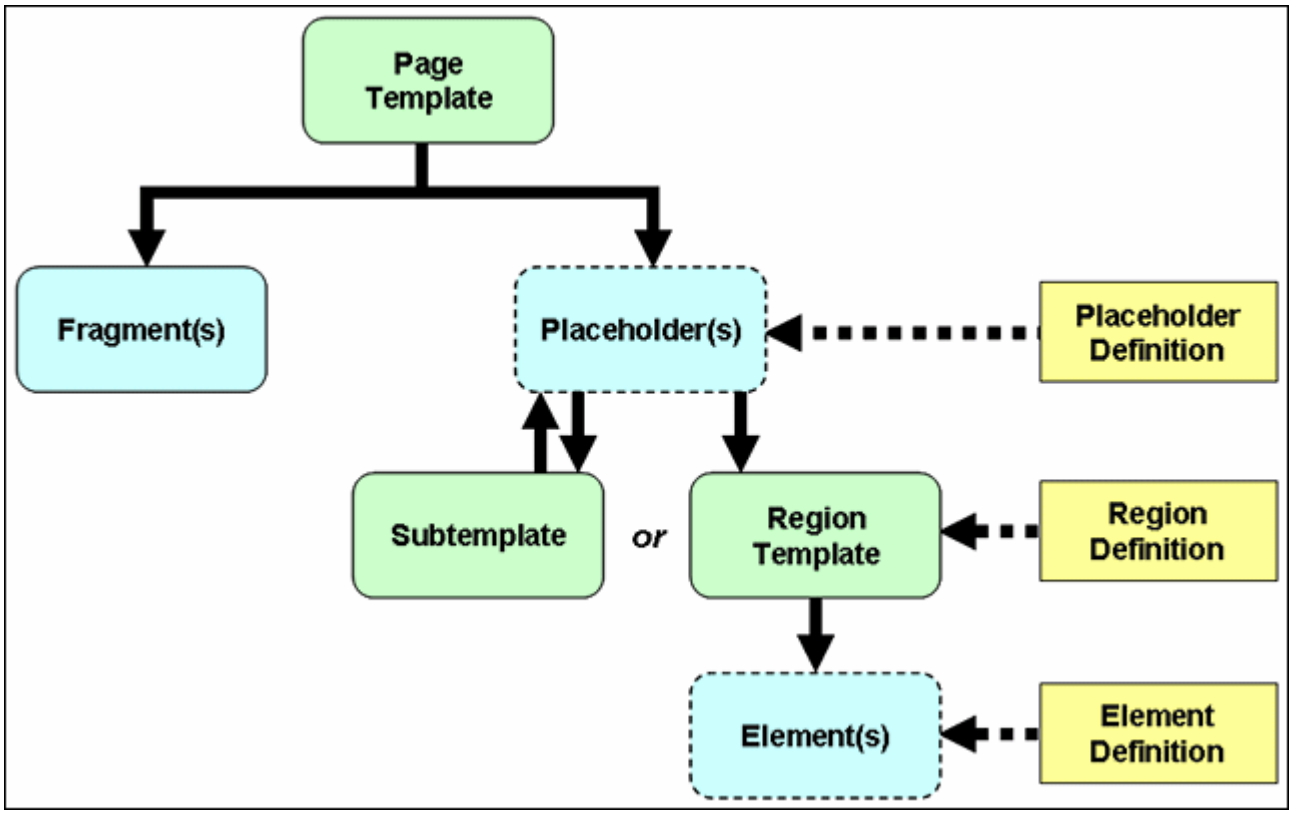
When the contributor is done editing and clicks the save icon in the Contributor editor, the data file is updated and checked into the content server again. The web page is then updated in accordance with the site update schedule.

Please note that the Contributor editor always shows *all* elements in the region definition (and hence, data file) associated with a contribution region, even if they are not used in that particular region. The other elements may be used elsewhere on the web site, so editing that information may affect other pages on the site.

2.6 Site Object Hierarchy

Figure 2–4 shows the hierarchy of site objects that are used to create and manage a Site Studio web site.

Figure 2–4 Object Hierarchy of Web Sites



Page templates are at the top of the hierarchy. They provide the framework for the pages in a web site within which the site content is displayed. In addition to standard HTML layout and formatting code, they contain site-wide images and other assets,

and tags for fragments and/or placeholders. Page templates are stored and managed on the content server. For more information, see Section 2.12, "Page Templates."

A **placeholder** is an insertion point on a page template to identify where there is a contribution region (that is, editable area) on the web page. What that contribution region contains and what it looks like is defined using region templates and region definitions. A page template may contain multiple placeholders. There are no files associated with placeholders. Placeholders are controlled by *placeholder definitions*, which specify what content can go in the contribution region and how it is displayed, as well the actions available to contributors (for example, switching content or modifying metadata). A placeholder contains either one subtemplate or one region template. For more information, see Section 2.10, "Placeholders and Placeholder Definitions."

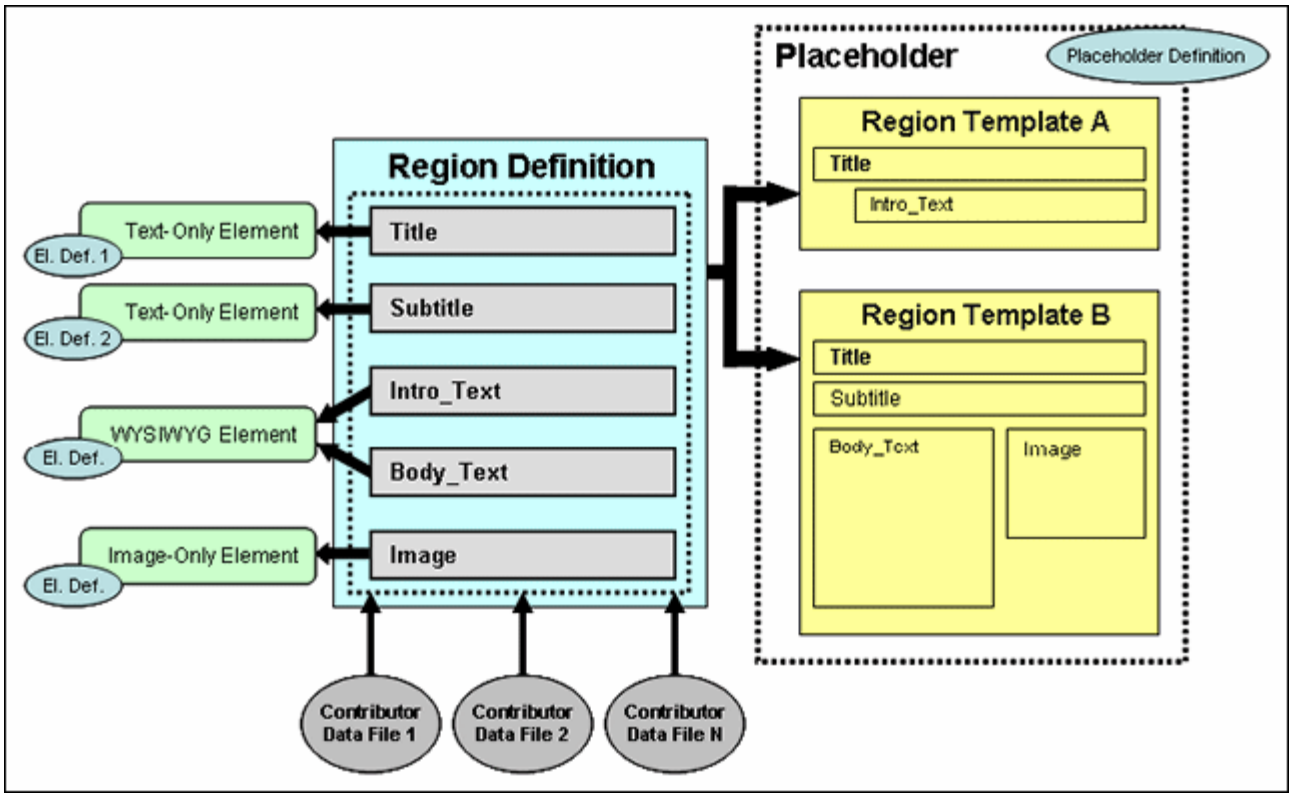
A **subtemplate** is a partial HTML file (that is, without a head and body section) that provides a mechanism to divide a placeholder on a page template into further smaller, reusable areas with their own placeholder(s). There is a circular relationship between placeholders and subtemplate; that is, a placeholder may contain a subtemplate, which, in turn, may include one or more placeholders. Subtemplates are stored and managed on the content server. For more information, see Section 2.11, "Subtemplates."

A **region template** is a partial HTML file (that is, without a head and body section) that defines the layout and look-and-feel of the data in a contribution region (marked on a page template using a placeholder tag). Region templates are controlled by *region definitions*, which define what kind of content can go in the region template. They also specify the content creation and switching options available to contributors for the contribution region, and set default metadata for content files associated with the region. Both region templates and region definitions are stored and managed as separate assets on the content server. A region template may have one or more references to elements. For more information, see Section 2.9, "Region Templates and Region Definitions."

Elements are the smallest chunks of reusable information in a Site Studio web site. They are referenced in region templates, which causes their data to be pulled into the region template using the layout and presentation defined in the template. A region template may contain multiple element references. There are no files associated with elements. Groups of elements are arranged in region definitions, which specify site content types. Elements are controlled by *element definitions*, which specify the editing experience available to contributors for an element type. Specifically, they set the available editing features in the Contributor editor when a contributor is editing elements in a contributor data file. For more information, see Section 2.8, "Elements and Element Definitions."

Figure 2–5 shows an example of a site object hierarchy.

Figure 2–5 Example of Site Object Hierarchy



In Figure 2–5, there is a placeholder (that is, an editable contribution region on a page template) that has two region templates available to it (as specified in the placeholder definition). Region template A shows a limited data view, with just the title and a brief introductory text. Region template B presents a more elaborate data view, with the title, subtitle, body text, and an image. Either can be used for the placeholder, depending on the site context. Both region templates are associated with a region definition that has elements for each of the reusable chunks of information (Title, Subtitle, Intro_Text, Body_Text, and Image). Each of these elements is associated with an element definition.

The Title and Subtitle elements are of the same type (text only), but they have different element definitions, which means that the editing features available to contributors are different. The Intro_Text and Body_Text elements are both WYSIWYG elements, which typically means that contributors have a broad array of editing options available to them when editing these elements (for example, the ability to add tables or use advanced text formatting). The editing experience for contributors is the same for these elements.

One or more contributor data files are associated with the region definition, and ultimately with the contribution region. Their structure matches that of the region definition. They contain the same elements: Title, Subtitle, Intro_Text, Body_Text, and Image. When a contributor decides to edit the content in a contribution region, its associated contributor data file is loaded into the Contributor editor, which provides one editing area for each element in the data file. The editing features available in each area are set by the element definition.

2.7 Reusing Site Assets

One of Site Studio's most useful and powerful features is the ability to reuse site assets within a web site and even across multiple sites (providing these sites are all managed on the same content server). When an asset is changed once, it is changed everywhere that asset is used. It is no longer necessary to keep track of all instances of a piece of data to ensure that all of the web pages are updated. This applies both to files associated with site presentation and site content (see Section 2.1, "Separation of Site Presentation and Content"). Page templates, region templates, elements, and the like are most efficiently used when they are used multiple times. Similarly, the same content files can be displayed in different locations on a site, either completely or partially (different segments), to suit the context.

Figure 2–6 and Figure 2–7 show an example of site content being reused. In Figure 2–6, you can see a list of items consisting of a title, image, and subtitle. This information is taken from separate contributor data files. Each of these list items could be hyperlinked to open the full page as shown in Figure 2–7. In fact, this is the type of design that you see on many sites. That is, information is displayed on a primary page, and the full information is shown on a subsequent page when you click a link.

Figure 2–6 Web Page With Introductory Content

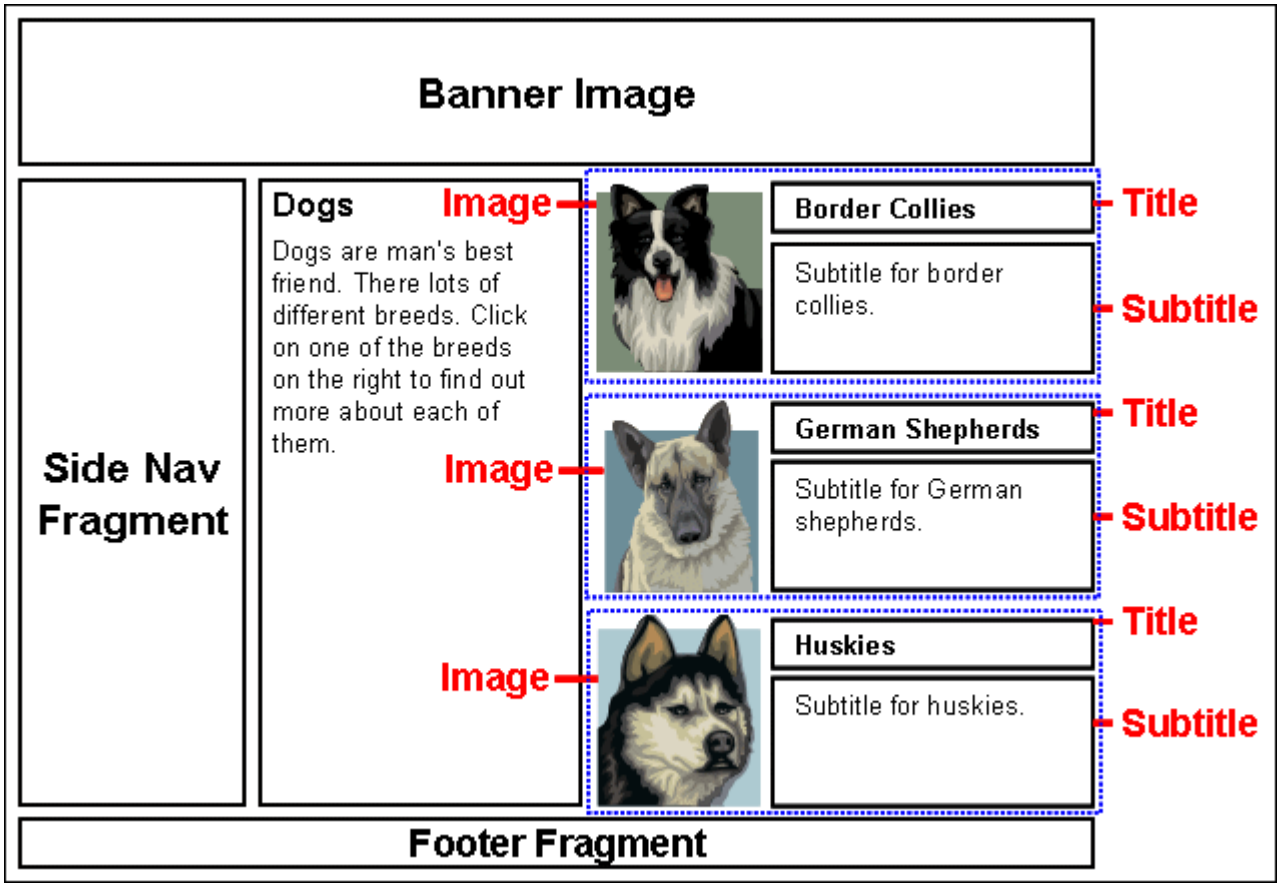
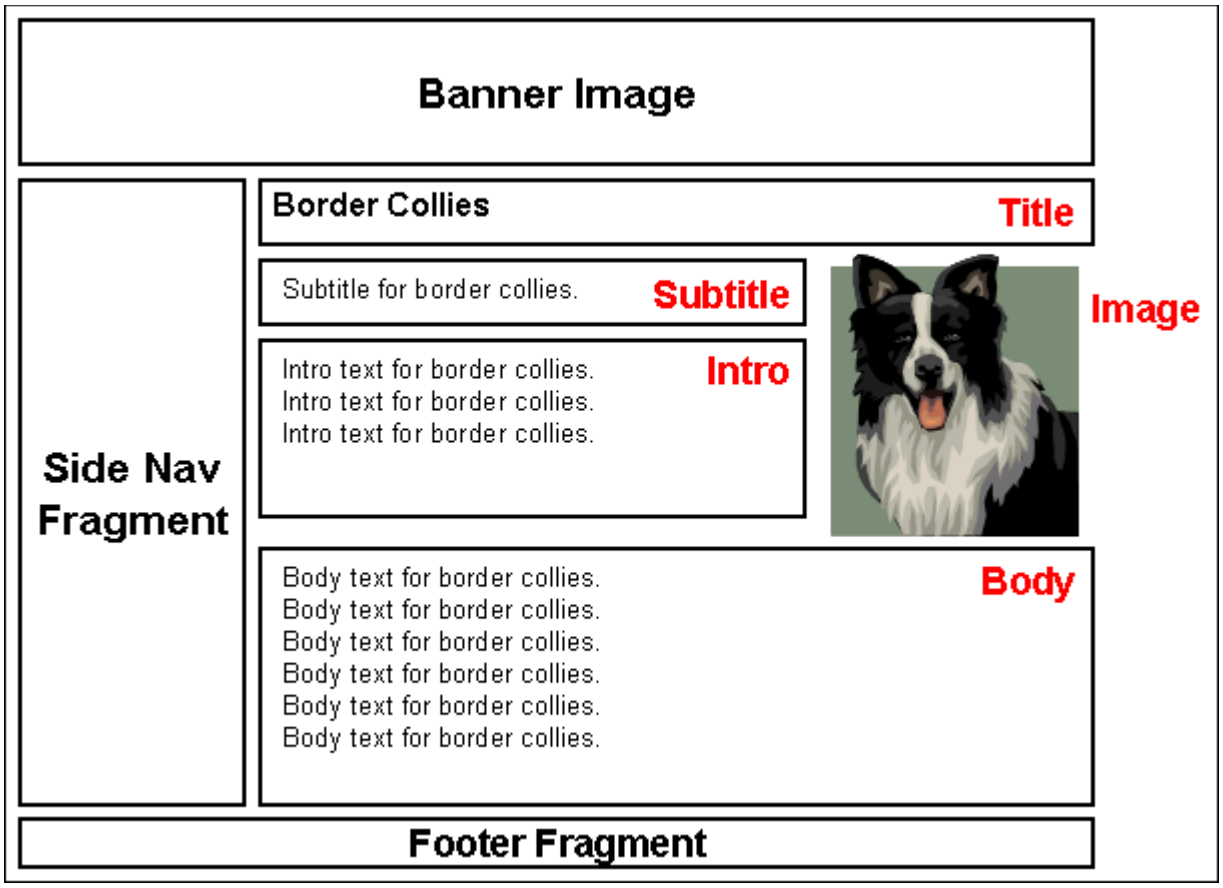


Figure 2–7 Web Page With Full Content



The web page in Figure 2–7 displays many of the same pieces of information. That is because the same data file is being used on both the region template in Figure 2–6 and the page in Figure 2–7. The elements are the same as well. An image-only element would be used to present the image, for instance, and a text-only element to present the title, with WYSIWYG possibly being the element type used to display the remaining information. There are few elements, typically, necessary in creating a web site, as one WYSIWYG element can be used anywhere on the web site where you may want that style of editing for the contributor.

Because site assets are intended to be reused across the web site, it is particularly important that the web site is fully planned out before the designer creates anything in the web site. For more information, see Chapter 3, "Planning Your Web Site."

2.8 Elements and Element Definitions

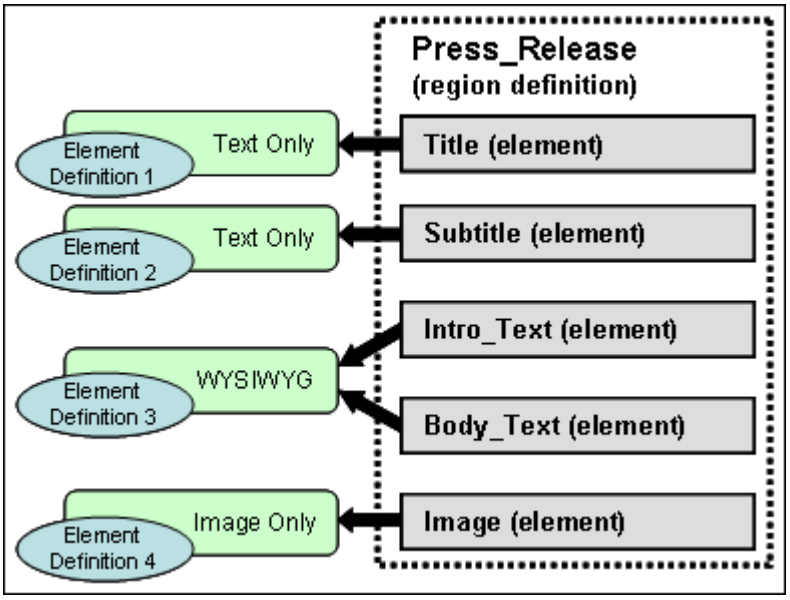
Elements are the smallest chunks of reusable information in a Site Studio web site (for example, a title, a product image, or the body text of a press release). Since element data can be reused within (or even between) web sites, you, as the site designer, should carefully consider how the site content should be broken up into segments. This may take some work up front, but maximizes reusability of site content and also makes managing the site content more efficient in the long run.

Each defined element is of a particular type: WYSIWYG, text only, image only, static list, dynamic list, or custom. These types characterize what the element content consists of, and, through element definitions, what editing options are available to contributors. For example, the title of a press release could be set up as a text-only

element (which typically provides only limited editing options to contributors), whereas the actual press release text could be a WYSIWYG element (which typically gives contributors much more editing control, such as the capability to add images or tables).

Elements are controlled by *element definitions*. An element is basically an instantiation of its associated element definition, which specifies the editing experience available to contributors for the element on the web page (specifically, what editing features are available in the Contributor editor). Element definitions are individually managed site assets, which means that they can be reused within a web site, or even across web sites (providing all sites are managed on the same content server). As shown in Figure 2–8, elements of the same type (Title and Subtitle) may have different element definitions associated with them to provide different editing environments for contributors depending on the context in which the element is used. Similarly, multiple elements (Intro_Text and Body_Text) may share the same element definition, providing the same editing experience to contributors for each of the elements.

Figure 2–8 Elements and Element Definitions



Individual elements are not separately managed site asset. Groups of elements are arranged in region definitions, which specify site content types. As shown in Figure 2–8, there could be a region definition called “Press_Release,” which consists of the elements Title, Subtitle, Intro_Text, Body_Text, and Image. A region definition could thus be thought of as a 'content class'.

Element data is stored in contributor data files associated with region definitions. Each contributor data file contains an instance of each element from its associated region definition. In the example of Figure 2–8, the contributor data files would have five elements called Title, Subtitle, Intro_Text, Body_Text, and Image. Contributors can edit the element data in the contributor data files using the editing features available to them for each element, as set in the associated element definitions.

2.9 Region Templates and Region Definitions

Region definitions define the types of content used on a web site. They could be thought of as 'content classes'. They are essentially groups of individual elements which define the various chunks of reusable information for a particular site content type. For example, as shown in Figure 2–8 on page 2-12, there could be a region definition ('content class') called “Press_Release,” which consists of the elements Title, Subtitle, Intro_Text, Body_Text, and Image. Contributor data files are associated with a region definition to store the data for each element in the region definition. (What a contributor can do with the data is controlled by element definitions; see Section 2.8, "Elements and Element Definitions".)

In addition to defining site content types in terms of its constituent parts (elements), region definitions also specify the content creation and switching options available to contributors for its associated contribution region(s). For example, if a contribution region is set up to allow contributors to switch the content of that region, they might be allowed to use existing contributor data files on the server only (not native documents or new contributor data files). (Please note that placeholder definitions control whether contributors can actually switch content in contribution regions.) Finally, region definitions also set the default metadata for content in contribution regions as it is checked into the content server.

Region templates define the layout and look-and-feel of the data in contribution regions within web pages. They are partial HTML files in that they do not have a head and body section. This allows them to be inserted into other HTML code as the web pages are generated for the Site Studio site.

Region templates consist of layout and formatting code, along with Site Studio tags to specify where elements (from contributor data files) or dynamic conversions (of native documents) are placed. Some elements from contributor data files may be displayed in some region templates, but not in others, which allows the information to be reused across different pages (as shown in Figure 2–5 on page 2-9).

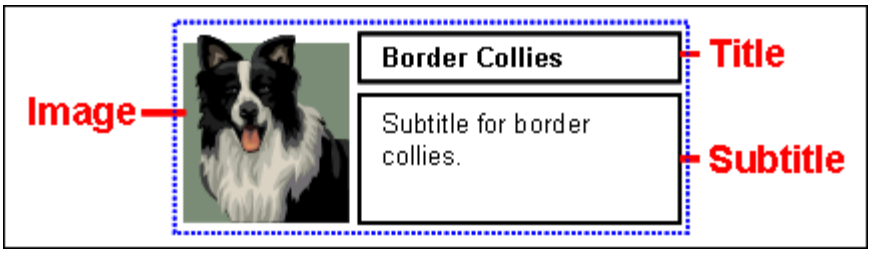
The site designer will probably create region templates more than any other site asset. Region templates allow you to present the information in contributor data files or native documents differently in various contexts of the web site. As with elements, it is worthwhile spending time considering how the information on your site should be presented through region templates. Judicious use of region templates maximizes the reusability of site content, and also makes site content management more efficient.

In short, region definitions specify what contribution regions on a web page contain, whereas region templates define what contribution regions look like. In other words, region definitions specify the structure (and attributes) of site content, and region templates define the visual presentation of that content on a web page.

There can be multiple region templates for each region definition. This allows site content to be displayed differently depending on the context within the site. If there are multiple region templates for a region definition, then the default region template is used unless a different one was specifically set to be used.

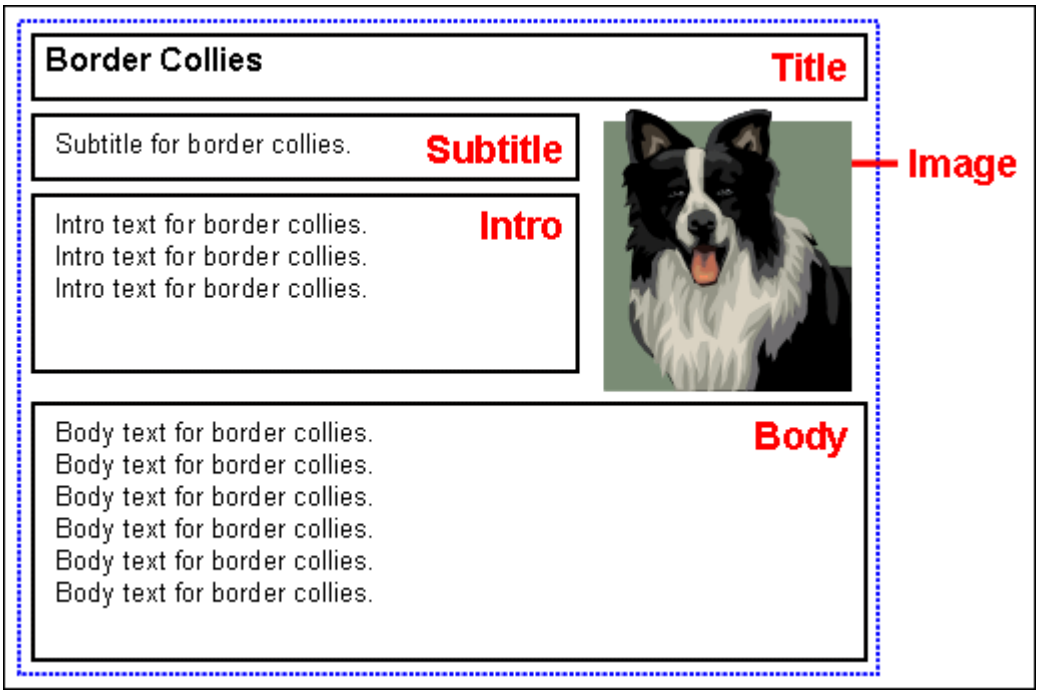
Region templates can be used to display information in multiple places in different layouts, while using data from the same data files. A common example of this is a list of items showing, for example, a title, a brief subtitle, and a small image. An example of this is shown in Figure 2–9, which represents a region template (with sample content) that shows these three elements arranged with the image on the left and the title and subtitle on the right.

Figure 2–9 Region Template With Limited Element Set From Region Definition



The content in Figure 2–9 could be hyperlinked to open a new page with a different region template that shows the same data, but now with more elements from the data file. Figure 2–10 shows an example of such a region template (with sample content), with a title at the top and a subtitle, introductory text, and body text below it. The region template also includes the image (although sized and positioned differently). All content is taken from the same data file as used in Figure 2–9.

Figure 2–10 Region Template With Full Element Set From Region Definition



In the case of Figure 2–9 and Figure 2–10, the contribution regions could be set up to allow contributors to edit the data on either page, but they would be editing the same data file and any changes would be reflected in all places where the data file is used. Please note that if contributors edited the data file from the region template showing the limited element set (Figure 2–9), they would see *all* elements of the data file in the Contributor editor, even though not all elements are being used in that particular context.

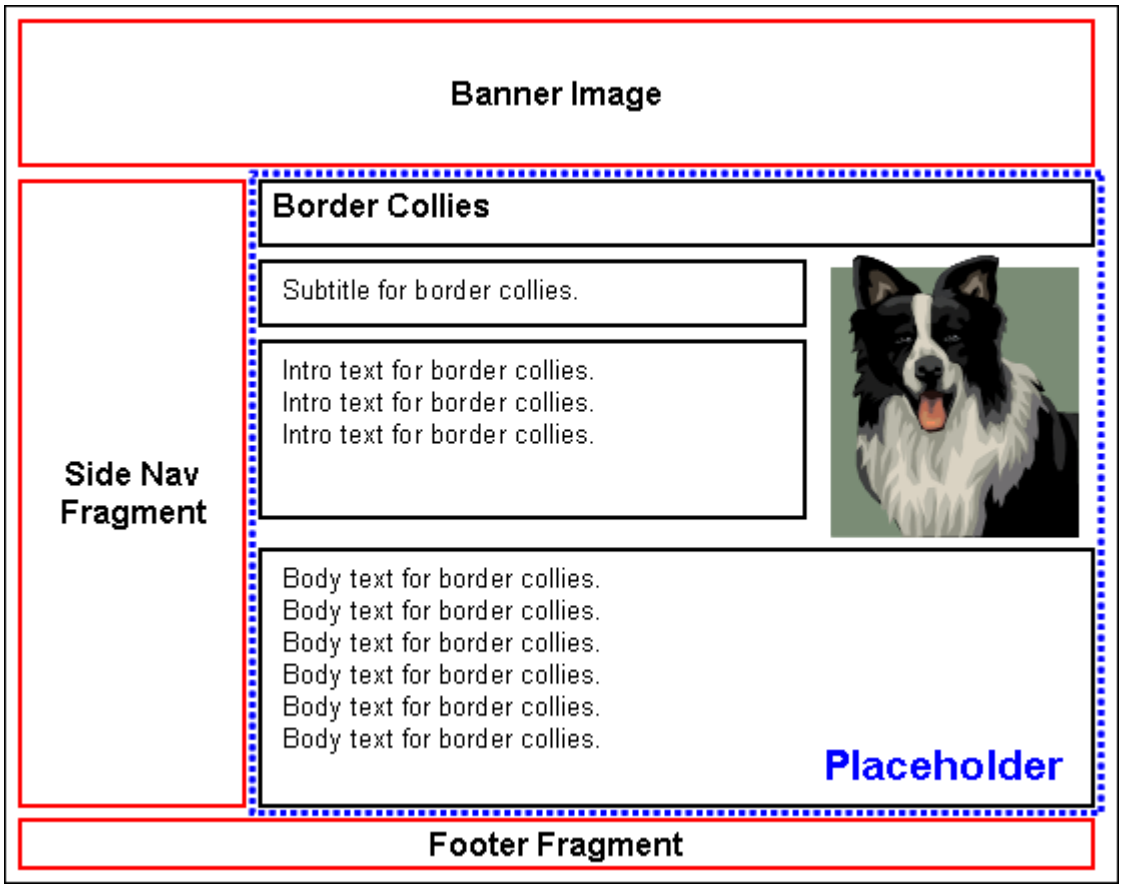
2.10 Placeholders and Placeholder Definitions

A Placeholder is an insertion point (a tag) on a page template (see Section 2.12, "Page Templates") to identify where there is a contribution region (that is, editable area) on the web page. A placeholder is represented by a simple tag:

```
<!--$wcmPlaceholder("Name")-->
```

What the contribution region identified by the placeholder contains, and how it looks on the site, is defined using region templates and region definitions (see Section 2.9, "Region Templates and Region Definitions"). A page template may contain multiple placeholders, each of which representing a contribution region on the page. There are no files associated with placeholders.Figure 2–11 shows a representation of a web page that contains one placeholder, marked with a dotted line (with sample content).

Figure 2–11 Placeholder on Web Page (With Sample Content)



When you insert a placeholder in a page template, all you are doing is marking a named position in the template where content may be inserted. To control how content is handled at that position, you must associate the placeholder name with a *placeholder definition*. Placeholder definitions specify what content can go in the contribution region and how it is displayed, and the actions available to contributors. For example, a placeholder may be set up to allow contributors to update the metadata of content displayed in the contribution region, or they may be allowed to switch the content of contribution regions. (Please note that region definitions control what kind of content contributors can switch to.)

Associating a placeholder with a placeholder definition (also called 'mapping') can be done in several ways:

- At the site level (using global mappings): You can set up default placeholder mappings that apply to the entire site unless any section-level or placeholder-level overrides are specified. You do this in Designer by opening the Tools menu and then choosing Define Placeholder Definition Mappings. This enables you to associate placeholder names with placeholder definitions for primary and secondary pages. The placeholder names are subsequently referenced in page templates. Placeholder names may be used on multiple templates in multiple sections and the mappings still apply.
- At the section level (using a section property): You can also set a specific placeholder definition mapping for the primary and/or secondary page of a site section. If you establish the mapping in this way, then it replaces any mapping that may have existed at the site-wide (global) level. Please note that the section where you do this now uses different mapping from any other section where the same placeholder is used.
- At the placeholder level (using a parameter in the placeholder tag): You can also set a specific placeholder definition for a specific placeholder. You do this by adding the `placeholderDefinitionDocName=[NAME]` to the placeholder tag (in source view). If you establish the mapping here, then this overrides all section-level and site-wide (global) mappings. It also means that this template uses this definition everywhere it used. The template then always uses the specified definition regardless of where in the site it is used. Also note that the only way to change this is to modify the template in source view. This would be considered the least flexible way of specifying a definition mapping as it is hard-coded, but this method can be used if desired.
- As a catch-all (using web site properties): You can set up a default placeholder definition that is used if none of the three other methods above apply. You do this by setting the Default Placeholder Definition property in the web site properties category of a site.

You can also set up a default placeholder definition, which serves as a catch-all placeholder (if no other applies).

Placeholder definitions specify what content can go in a contribution region (as marked by a placeholder tag) and how this content is displayed, as well the actions available to contributors. For example, a placeholder may be set up to allow contributors to update the metadata of content displayed in the contribution region, or they may be allowed to switch the content of contribution regions. (Please note that region definitions control what kind of content contributors can switch to.)

Placeholder definitions also specify what region definitions, region templates, and subtemplates are available for the associated placeholder (that is, contribution region) on the web page. Take, for example, a placeholder definition that allows three region definitions: REGION_DEFINITION_1, REGION_DEFINITION_2, and REGION_DEFINITION_3.

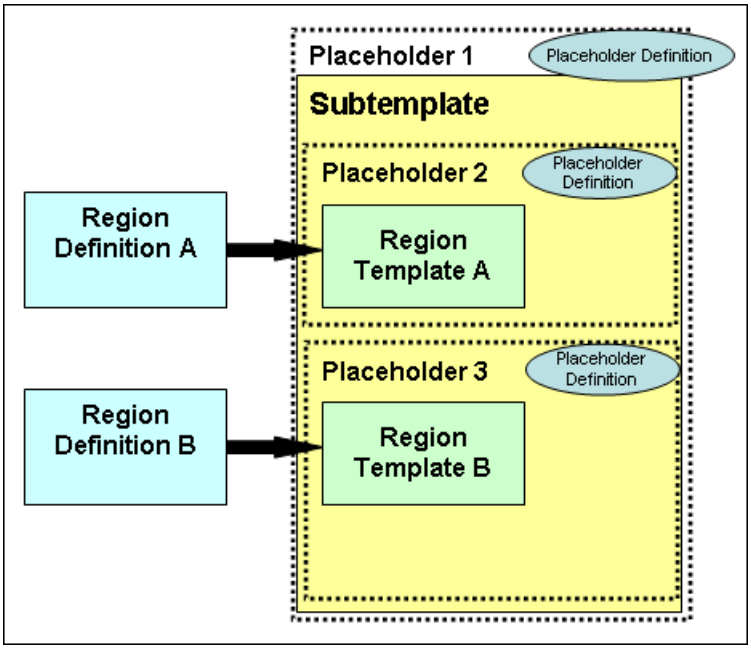
Region definition 1 has three associated region templates: A, B, and C, with region template A being the default one. This means that region template A is applied to all content files (contributor data files or native documents) associated with region definition 1 unless a different region template was specifically set. You generally associate a region definition with a content file when you create that file, although you can always change this association on the content information page (see Section 2.2, "Site Asset Storage").

2.11 Subtemplates

Subtemplates are the same as page templates, but do not have <HTML>, <HEAD>, and <BODY> sections. As such, they are essentially chunks of HTML code that can be inserted in page templates. Subtemplates may contain very simple HTML code, but they can also be quite complex, with their own scripts and the like. The code in a subtemplate is treated exactly as it would be when placed directly in a page template.

As shown in the site object hierarchy (Figure 2–4 on page 2-7), subtemplates can only be placed within placeholders, and subtemplates may contain their own placeholder(s). Subtemplates are typically used as a method of dividing a placeholder (that is, contribution region) on a page template into further smaller, reusable areas with their own placeholders, as shown in Figure 2–12. A placeholder can contain a subtemplate that contains one of more other placeholders, each with their own subtemplate or region template.

Figure 2–12 Subtemplate in a Placeholder



Please note that a good site design does not necessarily need subtemplates, and many web sites do not use them at all, since the addition of a subtemplate does mean that the designer has an additional type of site asset to manage.

Subtemplates can be used to help reduce the number of page templates used in a web site. This would be done by creating one main page template that can be used as broadly as possible for ease of reuse, and then in certain cases using subtemplates to change a placeholder on the main page template into multiple placeholders using subtemplates. This further allows reusability. A site designer can create a large area with one placeholder, which can then be used and reused with a placeholder containing a subtemplate with multiple placeholders in different layouts.

2.12 Page Templates

Page templates define the layout and high-level look-and-feel of web pages, including the placement of contribution regions (that is, editable areas on the page), navigation aids (in the form of fragments) and site-wide images (banners and the like). They provide the framework within which site content is displayed.

Page templates consist of layout and formatting code, along with Site Studio tags to specify where placeholders go. As such, they are typically light-weight in that they only contain high-level references to where contribution regions go on the page; they do not specify anything about what goes in these regions, both in terms of content and visual presentation (see Section 2.9, "Region Templates and Region Definitions"). Page templates typically include site-wide graphics, such as corporate banners or page layout images, and recurring, non-editable content, such as navigation aids or standard page content (footers and the like).

The number of page templates required for a web site depends on the site complexity, but usually only a very limited number should be necessary. For example, it could be sufficient to use one page template for the home page, and another page template for all other pages. The variations of content displayed on the various web site pages could then be handled using region templates (and region definitions) within the placeholders on the page templates. With fewer page templates, a web site is easier to maintain. This does require more work up front; that is, more thought must be put into the design of a web site, how the materials are laid out, and what information appears in which sections of the web site, and so forth. This also means that the site designer should carefully consider what information contributors are allowed to change, and what should remain fixed.

Figure 2–13 shows a generic page template that is often used for the homepage of a web site. It shows a banner image at the top, a navigation fragment on the left, a footer fragment at the bottom, and several small placeholders in the center, each of which representing some content. Placeholders do not delineate any page areas. They are no more than insertion points that specify the location of a contribution region. What goes in these regions and how it is presented is handled by region templates and region definitions. Site visitors can then click this content to see the content in full. This full content could be displayed in a page template that could potentially be used for all site pages other than the homepage. Figure 2–14 shows a generic example of such a page template. The template is basically same as for the homepage (Figure 2–13), but the content area is now a single placeholder to show page content in full.

Figure 2–13 Generic Page Template for the Home Page of a Web Site

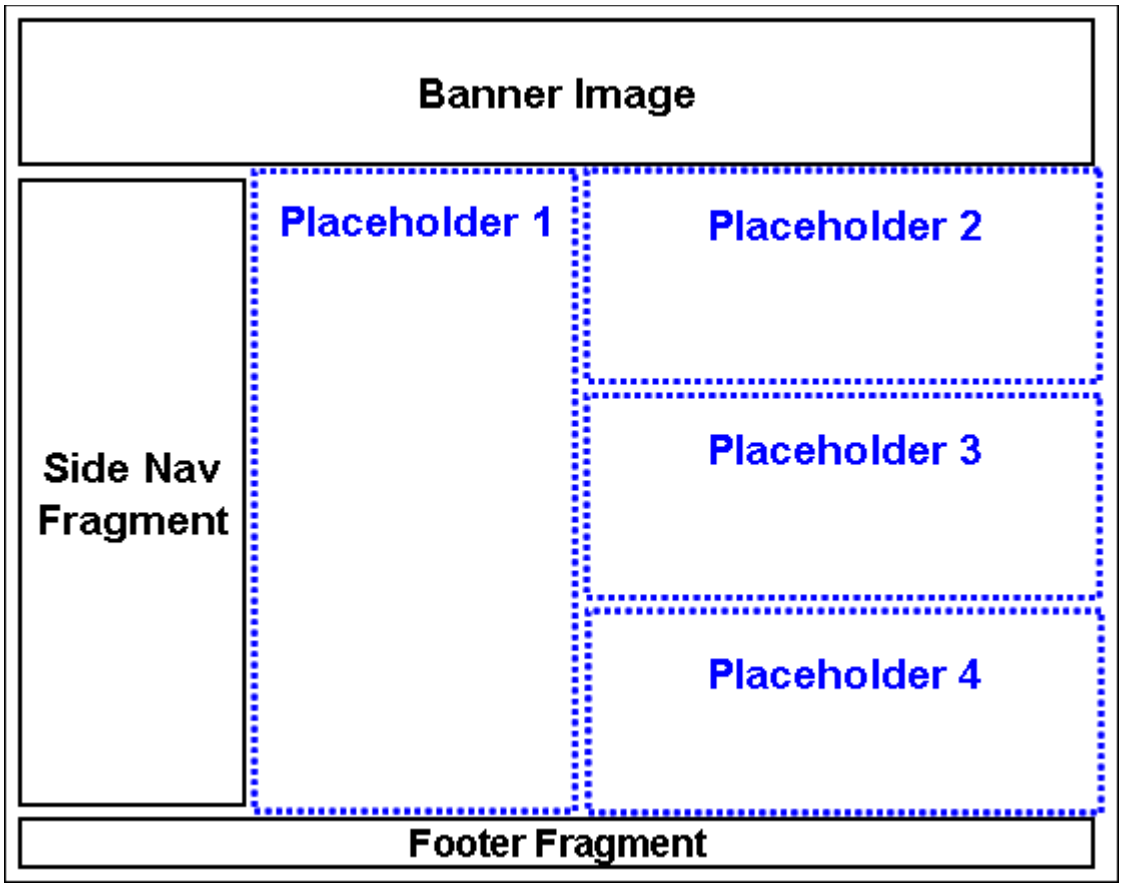
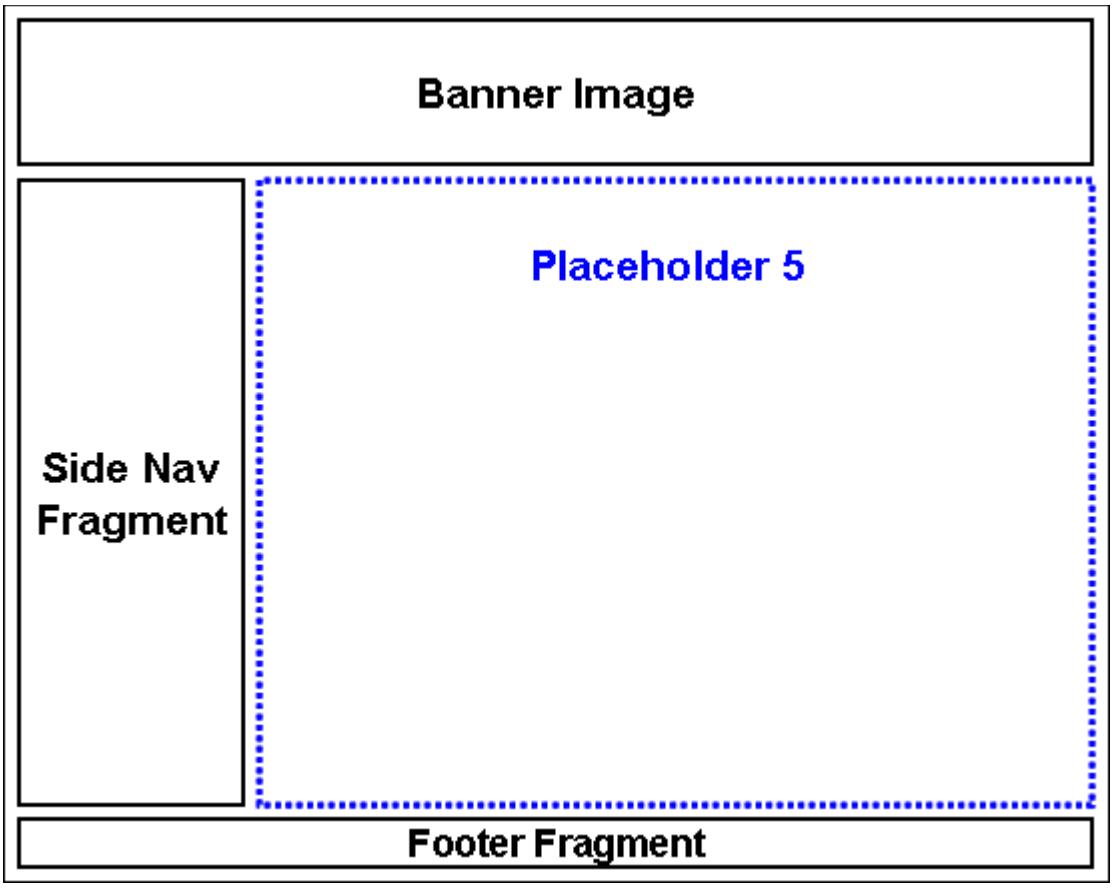


Figure 2–14 Generic Page Template for All Other Pages of a Web Site



2.13 Contributor Data Files and Native Documents

The content of Site Studio web sites is stored in either of two file types: contributor data files or native documents. These content files are stored on the content server, and they are the files that contributors interact with when adding or editing web site content. You, as the site designer, decide which of these (or both) are used for a particular web site.

The content files for a web site (contributor data files or native documents) are initially selected or created by the designer or contributor, and modified by the contributor. Contributors may be able to switch the content file associated with a contribution region if the designer has specifically allowed this in the placeholder definition associated with that region.

When a web page displays site content in a contribution region, it displays the content through a region template and region definition, which define where and how each named part of a content file displays within the page. You can assign a unique contributor data file or native document to contribution regions, or assign the same file many times, depending on whether you want to reuse specific content on your site. You choose the name and metadata for the content file when you first create it and check it into the content server. The names that you use for content files may be useful when managing these files on your site. Also, the names that you choose may depend on how often you are reusing content. For more information, see Section 3.4, "Planning and Naming Your Site Assets."

Contributor Data Files

Contributor data files are XML files that are created by Site Studio. Each contributor data file is associated with one (and only one) region definition that defines its 'content class' in terms of its constituent elements (see Section 2.9, "Region Templates and Region Definitions"). For example, if a region definition comprises four elements called Title, Subtitle, Body_Text, and Image, then all contributor data files associated with that region definition contain these same four elements. Each element in the contributor data file can be edited in accordance with the element definition associated with that element in the region definition (see Section 2.8, "Elements and Element Definitions"). The element definitions specify the editing options that are available to contributors when they edit an element in a contributor data file. Some elements may be set up to be plain text only with very limited formatting capability, while others might be WYSIWYG (What You See Is What You Get), which generally offers a much broader editing experience.

The content of contributor data files can easily be reused on a web site; that is, it can be displayed in different locations on a site, either completely or partially (different elements), depending on where it is used. It is worth noting that when contributors edit a contributor data file, they see *all* its elements in the Contributor editor, even if only some elements are actually displayed in the contribution region being edited. The other elements may be used elsewhere on the web site, so editing that information may affect other pages on the site.

Native Documents

Native documents are files that are created using familiar third-party applications such as Microsoft Word. They are converted to HTML format using Dynamic Converter so that they are viewable on the web site. Dynamic Converter uses conversion rules and templates to decide how to convert a native document. Native documents are edited using their associated application (for example, Microsoft Word for '.doc' files).

The content of native documents can also be reused on a web site, although native documents are generally not as flexible in their reusability as contributor data files. Judicious use of styles and other formatting features in Microsoft Word may overcome some reusability drawbacks of native documents. One important benefit of using native documents is that most contributors are already quite familiar with, say, Microsoft Word, which makes this application an easy and convenient editing environment for the site content.

2.14 Primary and Secondary Pages

The primary page of a site section is the page that is displayed when a visitor first enters that section. Sections in the site hierarchy usually have a primary page assigned to them, but this is not required. For example, a search results page, which you do not want users to browse to directly, may only have a secondary page. You assign a page template as the primary page of a site section.

The information on a primary page is statically linked. Contributors can change the contributor data file on the primary page using the Contributor editor, and native documents using their associated third-party application.

Secondary pages are optional for site sections, and they are typically used to dynamically present content on a web site. A secondary page can have static content, but what makes secondary pages useful is their ability to have dynamically placed and replaceable content. As such, they are used to create multiple versions of the pages within a site section; they provide a different content view for a site section. Secondary

pages allow you to handle large sites without needing to physically create thousands of pages.

A secondary page basically serves as the backdrop for content added to the site by a contributor. Secondary pages are required if you allow contributors to add new contributor data files or native documents (both of which amount to new web pages) to the web site. These files are made available to the site when they are picked up by a dynamic list, a search, or the target of a link. Page templates can be configured as secondary pages.

You can create a site comprised entirely of primary pages, but then you must create sections with new primary pages in order for the site to grow. By using secondary pages, your site can grow on its own from additional content submitted by contributors, and you, the designer, do not have to do anything. Your site becomes much more scalable with secondary pages.

One common use of secondary pages is with dynamic lists of hyperlinked items (for example, press release titles, as shown in Figure 2–15), each of which, when clicked, opens in full on a secondary page. More specifically, the target of the link opens in the replaceable region on the secondary page of the same section (by default).

Figure 2–15 Dynamic List With Press Releases

3

Planning Your Web Site

Planning is key to building a successful web site with Site Studio. Before you begin inserting text, graphics, and scripts into your page templates, you should ask “What is the function or role of the site?” Is it a department-level site, a company-wide site, an internal site, an external site? How many users will visit the site? How many users will contribute to the site? Will there be different security access levels for each contributor? Do you plan to replicate or publish the site? Is the site expected to grow over time?

These are all important questions to ask before you begin. We cannot predict your particular needs, but we can suggest some key points that you should consider before developing your site:

3.1 Why is Planning Important?

Proper planning of the web site helps determine how to maximize reusability of the web site through appropriate use and reuse of page templates, subtemplates, and region templates.

It is important to understand that the more time spent on the planning process, the easier the web site and the site assets are to create and manage. You might think that there is more time required to plan a managed web site using Site Studio, but the result is much less time spent managing the assets.

Proper planning is vital to making the site easier to run. In the beginning stages, it might seem as if more time is being spent before any pages are complete compared to older methods of creating a web site. But the results of time spent properly planning the web site makes the construction of it through the site assets much easier, and the maintenance of the web site, especially when making later changes when the web site is live.

What Parts of the Site Will Be Reused?

When you consider the web site, you should look at all content and all of the structure and consider what is reusable, and what should be used only once. When considering this, it could be thinking of simply the layout of the page, or it could be simply what data is displayed, or it could be a consideration of a certain piece of data displaying in a certain way.

Because there are so many ways of arranging and reusing the different parts of the site, it may be helpful to look at these examples of organization and reuse to think about while you consider your own web site.

In a typical web site, there is the navigation on the left, the banner graphic on top, and a large central area with the information on the page itself. We would expect that the banner and the navigation should be on all pages, so this would be placed on the page template itself. But the information in the middle will obviously be different from page to page. This is where the considerations are most important.

The way the information is organized is the most important consideration. When you look at one page, it may have objects arranged in one column, or in an array, or broken up with images. It's possible to arrange everything in one placeholder, but there is the other aspect, where you can create smaller sections, each with its own smaller contributor data file.

Consider a page on your web site that would list open employment positions. You could create the page such that it is one placeholder, listing all internal positions and all external positions. Or you could create a subtemplate within that placeholder (which would then contain separate placeholders and region templates, and so forth), so that the external positions would be stored separately from the internal positions. Each could be maintained in a separate contributor data file, so that the external web site would contain only the external announcements, and the internal web site would contain both the contributor data file with external announcements and the one with internal announcements.

Another use would be where each department in the company could list their own job openings; then, on one central page, you could collect all of those individual openings and display them all. In these instances, you can use a subtemplate to easily manage the differing numbers of placeholders.

Other considerations for how you lay the data out on the page, and how to organize the placement of the data within the web site, needs this kind of consideration on a page by page basis.

You should consider these questions: Would it be best to use one placeholder on the page template, then use a subtemplate to break that placeholder up into parts? Or would it be better to have a few more page templates to allow for different placeholder arrangements?

Another example would be an instance where you have a small piece of information that does not necessarily need a separate page, but you would definitely want to reuse. An example of this could be stockholder contact information, or possibly job application information, separate from typical corporate contact information. The information is not enough to necessarily warrant its own page.

In all of these cases, the page template would be the same. It would have the banner, the navigation, a footer, and then in the middle, the placeholder representing the area that can be replaced and filled with any information you need, structured exactly as you need it. It was the consideration of how to use a subtemplate to further use a placeholder or multiple placeholders within that template that enables you to keep the single look that you need for all pages.

It would also be possible to achieve this layout with different page templates on each page. Again, it depends on how you plan your site.

As you can see, the most important part of the site creation is to figure out how each portion of the web site, both in terms of structure and content, is displayed. With Site Studio, the more time you invest in planning before you create, the less time you spend creating the hundreds and even thousands of pages your web site delivers.

3.2 Planning Your Site Hierarchy

The site hierarchy is the framework of your web site. You should give yourself plenty of time to plan the hierarchy before you start creating web pages. The site hierarchy not only helps you organize and manage content on your site, but it is also used by Site Studio to automate certain tasks.

How Deep Should the Hierarchy Be?

A deep hierarchy places information at multiple levels where information is heavily categorized. This works well for large organizations or any organization that anticipates growth on the web site.

When planning the web site, a lot of care should be taken in considering the hierarchy of the web pages within the site. The web site hierarchy can be as deep as you want, and as deep as you want. You can create as many different sections from the home page as needed, and a section can contain as many sections as needed as well. However, the thing that should be kept in mind while designing the hierarchy is that while the structure can be as wide as needed (that is, sections can be nested to any depth), that this can create unwieldy URLs. The wider a section is nested in a web site, then the longer the URL is to retrieve that information. Usually this is a trivial consideration, but for some designers this can be a major point.

Each section listed in the site hierarchy can have a primary page, and a secondary page. Since secondary pages are the pages that have replaceable content, the secondary pages are used to create multiple versions of the pages within a section. The primary page within a section is the page that opens for that section, it could be considered the landing page for that section.

How Will The User Navigate the Site?

While you use your site hierarchy to manage your site, visitors use your hierarchy to browse to and locate content. In Site Studio, when you add a navigation fragment to a page template, the fragment reads your site hierarchy and generates links that comprise the overall site navigation. You can easily add, remove, and rename sections of your site, and these changes are seen in your site navigation. You should think carefully about the visitor's experience as you construct this hierarchy.

How Should the Sections Be Named?

Your site hierarchy has individual sections with names such as Products, Services, and About Us. These names are important. They not only help you organize content on your site, but they display in the navigation on the site, where visitors see it. It is a good idea to revisit these names regularly to ensure that they reflect the content of the sections they represent.

Another thing to consider as you assemble your site hierarchy is the web site address that contributors and site visitors see in their web browser address bar. By default, Site Studio uses the names (labels) that you give to the sections in your site hierarchy.

You can override these values, if you like, by specifying a different path name and page name for each section. It is best to plan this ahead of time to minimize any late changes to the site address or paths used in the address, which could result in broken links or missing shortcuts for contributors and site visitors.

How Reusable Should The Page Templates Be?

As you plan your site, you can create new page templates for each section, or you can reuse page templates that are already being used in other sections. When you design to

use a small number of page templates, you then have fewer assets to manage, which makes the entire web site easier to manage.

By reusing page templates, you can maintain the look and feel of your web site in one or just a handful of page templates, and the changes are seen immediately across the site.

By creating new and unique page templates in each section, you might have more flexibility in the design of the site layout as a whole. You can create a unique design or slightly modify a different page template to accommodate the content that display there. However, if you make significant global changes to the site, then you have to modify multiple page templates. This might become a repetitive task that lends itself to mistakes and inconsistency.

If you choose multiple page templates, you should try to reuse fragments with referenced snippets as much as possible so that you can make global changes to just one or a handful of fragments.

However, through thoughtful use of subtemplates with the placeholders, it is typically possible to reduce the number of page templates.

Should There Be Both Primary and Secondary Pages?

There are several ways to handle page templates. You can assign one primary page and one secondary page for every section of your site hierarchy.

The primary page is the web page that users see when they go to that section on the web site (similar to the default page on a conventional web site). The information on the page is statically linked. Contributors can change the contributor data file or the native document on the primary page, or the information in the data file through the contributor interface.

The secondary page is a page that can have its content dynamically placed. A secondary page can have static content, but what makes secondary pages useful is their ability to have dynamically-placed content.

You can create a site comprised entirely of primary pages, but then you'll have to create new sections with new primary pages in order for the site to grow. By using secondary pages, your site can grow on its own from additional content submitted by contributors, and you, the designer, do not have to do anything. Your site becomes much more scalable with secondary pages.

Primary and secondary pages can have the same page template. The page template is not related to whether the page using the template is primary or secondary.

How Will Content Be Reused?

You can share and reuse contributor data files and native documents across a single web site and even multiple web sites in the content server. When you add these items to your site, you specify a specific location, or multiple locations by reusing associated site assets, where you would like them to appear.

You can even add content to your site that is not currently associated with any web site in the content server. You can show the same document on different sites, each with its own look and feel, without having to create multiple versions of that same document.

If you choose to do this, however, you must ensure that the content is appropriate for the web site and for public viewing. Workflows are good to implement to ensure this. You also must ensure that you have a secondary page with a replaceable region template in each section where the content appears.

Will a Manager Be Necessary?

You can create and manage the site hierarchy yourself, or you can hand this responsibility over to a site manager. The site manager can create sections, modify sections, and more.

These changes can significantly change the appearance and behavior of the site. You should think about these changes and how they affect the structure of the site, workflow, replication, the role of contributors, and much more.

3.3 Planning Your Contribution Model

As you plan the site hierarchy and prepare the assets for the web site, you must think about your contribution model. That is, how content gets placed on the site by users (contributors and managers).

How Much of the Site Will be Provided by the User?

There are some things that the designer must add to the site, such as the background color, positioning devices (HTML tables or CSS), site navigation, and custom scripts. But much of the content for each web page (that is, the actual information on the page) can be created and edited by a contributor.

To really harness the power of Site Studio, you should open up as much of the web site as possible to these users. This way, your web site can be continuously updated without the bottlenecks or delays typically associated with a web site.

How Much Control Will the User Have?

On every page template, you can have one large contribution region or several small contribution regions. Within a region template, you can have one or several elements, each one appearing as a field where users (contributors) add and edit content.

There are different types of elements that can be used for specific purposes, like adding and editing text, graphics, and updating lists. You can turn on or off certain formatting attributes, such as the choice of typeface, font size, images, tables, and custom properties. These choices depend on how much control you want the contributor to have over a given web page. You can also enforce what content is added using the validation feature or your own validation scripts.

Will There Be Different Security Access for Each User?

You should know your users, their role in the organization, and their knowledge of web publishing before you give them access to the site. You may want to provide equal access to all contribution regions on the web site for all users, or you may want to limit access for some and grant full access to others.

For example, you may want to provide full access to the entire web page for members of the marketing department and access to only a portion of that page to all other departments. To do this, you would set up different users on the content server, assign a different contributor data file or native document to each region, and assign unique security metadata (using the metadata framework on the content server) to those files.

As a result, only the files that a particular contributor has permission to edit will display a contribution graphic (Figure 3–1) on them on the web page when in contribution mode.

Figure 3–1 Contribution Graphic

Sections of the web site can also be marked so that only certain users can view them. This is handled in the section settings within the web site hierarchy. After the section has limited access, the section is available for viewing (and editing, if you want) to only those users that you want to have access.

Will Contributors Submit Native Documents?

In addition to adding and editing content stored in Site Studio data files, contributors can also add native documents (created from applications like Microsoft Word, Excel, or PowerPoint) to the site. These documents can be converted into web-viewable renditions using dynamic converter component on the content server.

Contributors can add new and existing native documents directly to the web site (for example, by assigning one to a region, using the link wizard, or adding one to a dynamic list).

If you allow contributors to submit native documents to the web site (and again, you control this in a contribution region, an element, and a list fragment), you should educate them on how they should use the native document and any style guidelines you may be enforcing.

How Will the Contribution Process Be Coordinated?

If you are the designer, manager, and contributor of your web site, you know when and where the site must be updated. If, as the designer, you are working with a single contributor or manager, you must communicate the required updates for the site. Specifically, you should inform them of certain changes to the site. The contributor, however, may also alert you when content has been added.

If you are working with multiple users, communication can quickly become complex. You must include some kind of communication process between users. One solution is to introduce a formal review process using workflows. This means that various persons review and approve site changes before they become live.

3.4 Planning and Naming Your Site Assets

Site assets in Site Studio are intended to be reused. Since the asset itself can be used in not only different areas of a web site, but also in different web sites that you are designing, the naming of each asset is important. With a good naming scheme, assets can be more easily managed and more easily deployed in one or many areas of a web site, and one or many web sites.

The best way to name an asset is to describe what it does and how it functions, rather than by using a name based on where an asset is used in a web site.

The first thought of many designers is to put the name of the web site in the name of each asset. This would group them when listing the contents of the content server. But the difficulty here is when you run multiple web sites, since all of the assets are reusable. Rather than create an entirely new set of assets and definitions for a certain section in your web site, you can easily import the other assets you've already created. In doing so, though, when the assets have the name of the other web site in the name, this can cause some issues of continuity, understanding, and even some confusion regarding why some assets are present.

Names of the assets should also avoid the place of the asset within the web site as you may run into some issues when reusing assets. For example, by naming a page template originally intended to be the primary page template as *primary_template*, there may be some confusion when that same template also ends up being used on the secondary pages. Proper planning would have eliminated this issue, giving the template a name such as *page_template_with_2_placeholders*.

Consider these two different methods of naming the site assets in your project:

Naming convention with little planning toward maintenance:	Naming convention with considered planning toward maintenance
placeholder_def_1	element_def_WYSIWYG_fulledit
p_def_2	element_def_imageonly_minedit
default_definition	placeholderdef_subtemplate_newslist
front_page_template	regiondef_customform
subtemplate_up	regiontemplate_title_and_body
element_title	pagetemplate_home
region_replace	pagetemplate_errorhandler

Future updates are much easier to handle when the assets are named to describe what they do, allowing you to place them as needed. This also means you do not have to look in all managed web sites for an asset that you have already created.

3.5 Creating Your Site Assets

After the web site has been planned, the assets used to construct the web site should be created. For the most efficient use, it is recommended that you create these assets in a particular order.

The most efficient method of asset creation for most web sites is to build the assets from the bottom up, starting with your element definitions.

- Step 1: Creating Your Element Definitions
- Step 2: Creating Your Region Definitions
- Step 3: Creating Your Region Templates
- Step 4: Creating Your Subtemplates
- Step 5: Creating Your Placeholder Definitions
- Step 6: Creating Your Page Templates

It is not required that you create the assets in this order. However, your web site can be quickly and efficiently assembled if you follow this order.

3.5.1 Creating Your Element Definitions

The most efficient assets to define first are the elements. Elements are the assets that define the editing interface that the contributor uses. The specific use of each one is handled through element definitions. By creating an element definition, you are defining how the contributors work with the data files by defining the editing regions and the toolbars accessible to the contributors.

In the element definition you can control exactly which editing features are available to the contributor in the toolbar. For instance, you might not want the contributor to be

able to change the font size or color, but still be able to use bold and italics. Generally you define at least one of each kind of element (WYSIWYG, text only, image only, static list, dynamic list, and custom) and usually, you define two or more of each type of element to allow for the different toolbars in Contributor.

The element definition is used to control the toolbar that the contributors can access to edit the pages. However, it might be that you want to have several different definitions for each element so that you can have multiple options of toolbars for the contributors, rather than one or two. It is important to consider the role of the contributor and how much they contribute on different pages.

It is highly recommended that you name the element definitions carefully, as naming the elements with respect to place on the page or with the name of the site in them can make reusing them less intuitive. It is not recommended that you name the element in terms of its place on the page, within the web site, or even by using the name in the web site within the element definition. The optimal method for naming an element definition is to list a relative amount of access in the toolbar, such as *element_WYSIWYG_fulledit* or *element_text_undo_only*.

Notice that the element definitions were named starting with the word *element*. That is to make all elements appear together when searching for them in the content server. Using other naming conventions such as including the name of the web site would also group the assets in the web server, but this would make the definitions less intuitive to manage if you use the same definitions for multiple web sites.

3.5.2 Creating Your Region Definitions

You should create region definitions before you create region templates. A definition (region definition, placeholder definition, element definition) controls which data is displayed and how the contributor interacts with the data. The template (region template, subtemplate) control how the data is displayed. The region definition, specifically, controls which elements (WYSIWYG, Image only, and so forth) open in Contributor. The region template is used to arrange the placement of the elements and how that particular data is displayed on the web page.

As you create a region template, you are asked to associate it with a region definition. If the definition is not already defined, the region template is not easily associated with the definition afterward.

The typical region definition contains multiple elements, and various combinations of these elements are used on several different region templates. That is, the region definitions are as broad and encompassing as possible. This allows multiple region templates to access the same grouping of elements (but it does not have to be the same data file) and arranging them in different ways.

Because the region definitions limit the availability of elements in the contribution model, it is especially important to follow good naming conventions for region definitions. This helps you sort out which definitions are best suited for the different region templates you create.

3.5.3 Creating Your Region Templates

Region templates are the assets that format the data files into HTML. You use region templates to place the content, such as text, images, or other simple items, that would not be inside of an element, and the elements. The region templates are used to arrange the elements (limited by the region definition) into the layout desired to be reused as you see fit in the different places in your web site.

The region template has available to it only those elements that have element definitions listed in the region definition for the template. However, the region template does not have to use all elements listed in the definition. This allows for a great deal of variety in the different templates available.

It is likely that region templates are the largest number of assets you create for the web site. Because of this, the naming convention you use in creating the region templates is important. The name should help define what the template does, not where on the page it is used or where in the web site. A name such as *regtem_titlebodyview* or *rt_bodynoimage* would likely be a more useful name when maintaining the site than would, say *regiontemplate14* or *RT_SupportForum*.

Since region templates can contain HTML and images and other assets (for example, a region template-specific CSS) that are not part of the data file that is passed through the elements, the templates can be used to incorporate some information and text that is not editable by the contributor. However, it is important to understand that any of that information is not part of the contributor data file that the contributor edits. This means that as you reuse a region template, the data associated with it displays in the same manner regardless of what data is associated with that region template. It also means that the designer is the only person who would be able to modify that particular data. The contributor would be able to create and edit any data that is assigned to the elements in the region template through the placeholder, but anything that is placed outside of the elements, but still within the region template, would be static.

3.5.4 Creating Your Subtemplates

Subtemplates are major portions of HTML that can contain placeholders. This makes them very useful as a way to reduce several page templates while still maintaining several different ways the pages in the web site can appear.

Subtemplates can only appear in a placeholder. Subtemplates can only contain a placeholder, or static content such as HTML or an image. A subtemplate can also reference a CSS file.

Is a Subtemplate Necessary for the Site?

One of the benefits of using a subtemplate is that you can reduce the number of page templates used in a web site. Through a subtemplate, you can have the same page template used for both primary and secondary pages.

This is one reason why planning is essential. While a subtemplate can reduce the number of page templates, it also means that you have a subtemplate to manage for the web site. Fewer page templates means that site-wide changes are easier to make. Subtemplates can also add a level of static information because you can add HTML, images, and other similar static site assets to a page that you may want placed on a certain number of pages (such as the secondary pages) that would otherwise have to be managed separately from the subtemplate.

The subtemplate can also help you place multiple placeholders within a single placeholder. Since a placeholder can contain a subtemplate, and a subtemplate can contain a placeholders, then it could be that you want the subtemplate present for future web site expansion that is still to be determined.

3.5.5 Creating Your Placeholder Definitions

Placeholder definitions are what connect each placeholder to the content and other assets associated with that placeholder to display.

How Will The Placeholder Function On The Page?

The placeholder, not to be confused with the placeholder definition, is simply a mark on the page in the application to identify where there is a contribution region (that is, editable area) on the web page. Placeholders are simply conceptual boundaries. The actual amount of space that a placeholder uses on a web page is determined by the size of the content placed in it.

The data associated with the placeholder depends on which page the placeholder is on, which happens after the site structure is complete and you begin to place content in the sections of the web site.

It is the placeholder definition that determines how the placeholder functions, and what the placeholder contains. This means that you should view the placeholder as an available section for you to use and reuse content in broad areas. It is common to have a page template, for instance, with only the navigation and the header image and nothing but a placeholder in between. This placeholder would then be easily replaced on any number of pages with the content specific to those pages. The content can be broken down into smaller placeholders if your design planning has led you to create a site based on a minimal number of page templates and using multiple placeholders and subtemplates in different ways.

What Will The Placeholder Definition Control?

The placeholder is little more than a tag that is meant to define an area on the page where the content is shown. The placeholder definition lists which region definitions (and their associated region templates) and which subtemplates are available to pass information through.

You can also control other details on a placeholder-by-placeholder basis through the placeholder definition. Some of these items include workflow, whether the contributor can edit the data displayed through the placeholder, whether the contributor can edit the metadata, ability to view web site usage reports, and the ability to view content tracker reports.

3.5.6 Creating Your Page Templates

A large number of web sites can be reduced to two simple page templates: the home page, and then all other pages. It is not uncommon for a web site to have a home page that is distinct in terms of layout and design compared to the other pages in the web site.

It may be the case that there are multiple templates for your site. It is also possible to make an entire web site work with one page template. If you do design the web site with several page templates, you should possibly consider the use of subtemplates to reduce the number of page templates. Fewer page templates on a web site makes site-wide changes much easier.

Will Primary and Secondary Pages Require Different Templates?

Primary and secondary pages can both use the same page templates. However, since secondary pages are the only pages that can have dynamically placed content, you should consider the effect on how you view your page templates (and even your placeholders and placeholder definitions) with respect to the advantages of dynamically placed content.

A secondary page serves as the backdrop for content added to the site by a contributor. Secondary pages are required if you allow contributors to add contributor data files or native documents (both of which amount to new web pages) to the web site. These files are made available to the site when they are picked up by a dynamic list, a search, or the target of a link.

It may be that you first build your site with just primary pages, saving secondary pages until after you set up contribution regions on the primary pages and know exactly what type of content contributors submit to the site. Then, you could add the secondary pages to handle this content.

Regardless of whether you use the same or different page templates for the primary and secondary pages in your web site, it is important that you name the page templates appropriately. This is the same for all other site assets in Site Studio. Say, a page template is used for both the primary and secondary pages. If the name of that page template was based on where it was initially placed in the site, the template name could lead to confusion when the site is expanded and the page template is being reused for secondary pages as well.

The most efficient naming of site assets, including page templates, should be based on how the page template is used. Naming conventions based on where the asset is used in a web site (for instance, *page_template_primarypage*) or based on in the order of creation (for instance, *pagetemplate3*) can make the assets harder to manage.

4

Building a Site Studio Web Site Tutorial

This section describes the steps for creating a Site Studio web site in the JDeveloper environment. This tutorial demonstrates how to create a simple web site with navigation. It covers the following topics:

- Section 4.1, "Creating a Site Studio Project and Connection"
- Section 4.2, "Creating a Sample Web Site"
- Section 4.3, "Creating Sidebar Links"
- Section 4.4, "Creating A Dynamic Conversion Page"
- Section 4.5, "Adding a Style Sheet"

4.1 Creating a Site Studio Project and Connection

These are the steps for creating a Site Studio project and content server connection:

- Step 1: Creating a New Application and Project
- Step 2: Creating a Content Server Connection
- Step 3: Adding the Connection to the Project

4.1.1 Creating a New Application and Project

Follow these steps to create a new application and new site studio project:

1. From the JDeveloper main menu, select **File**, and then **New**.
2. In the New Gallery dialog, expand the **General** category and click **Applications**.
3. From the Items list, select **Generic Application**.
4. Click **OK**.
5. In the Name Your Application step:
   - For Application Name, enter `SiteStudioDemo`.
   - For Application Package Prefix, enter `demo.SiteStudioDemo`.
6. Click **Next**.
7. In the Name Your Project step:
   - For Project Name, enter `SiteStudioDemo`.
   - For Available Technologies, select **Site Studio Technology** and click the shuttle button to transfer the selection to the Selected Technologies list.

If you intend to deploy this Site Studio project to a WebSphere application server, you must also select **ADF Library Web Application Support** and click the shuttle button to transfer the selection to the Selected Technologies list.

8. Click **Finish**.
9. From the JDeveloper main menu, select **File**, and then **Save All**.

4.1.2 Creating a Content Server Connection

Follow these steps to create a connection to the content server and add it to your Site Studio project:

1. If the Application Navigator is not already open, select **View**, and then **Application Navigator**.
2. Select the **Application Resources** panel.
3. Right-click **Connections**, select **New Connection**, and then **Site Studio**.
4. In the Create Content Server Connection dialog:
   - Enable the **Application Resources** option (if not already selected).
   - For Connection Name, enter the hostname of your content server (for example, developmentserver2 or developmentserver2.mycompany.com).
   - The Design Time URL field is auto-filled based on the connection name you provided. If you are not using default settings on your server you may need to edit this field.
   - The Run Time URL field is auto-filled based on the connection name you provided. If you are not using default settings on your server you may need to edit this field.
   - Enable the **Specify Credentials for this JDeveloper Session** option.
   - For Username, enter your content server administrative username.
   - For Password, enter your content server administrative password.
5. Click **Test Connection and Login**.

   If the connection fails, verify that you have the correct connection name and URL for the content server and have supplied valid login credentials.
6. Click **OK**.

4.1.3 Adding the Connection to the Project

Follow these steps to add a content server connection to your Site Studio project.

1. In the Application Navigator, select the **Applications Resources** panel.
2. Expand **Connections**, and then expand **Site Studio**.
3. Right-click the new content server connection, and select **Add to Project**.
4. In the Select Project dialog, select **SiteStudioDemo.jpr** from the list.
5. Click **OK**.
6. From the JDeveloper main menu, select **File** and then **Save All**.

4.2 Creating a Sample Web Site

These are the steps for creating a sample web site:

- Step 1: Creating a New Web Site in the Content Server
- Step 2: Specifying User Credentials for Contribution Mode
- Step 3: Editing the Web Application Deployment Descriptor
- Step 4: Creating the Home Page
- Step 5: Associating the Page Template with the Site
- Step 6: Running the Site and Viewing the Home Page
- Step 7: Creating Site Fragments
- Step 8: Creating the Element Definitions
- Step 9: Creating a Region Definition
- Step 10: Creating a Region Template
- Step 11: Creating a Placeholder Definition
- Step 12: Adding a Placeholder to the Home Page
- Step 13: Assigning Content to the Placeholder

4.2.1 Creating a New Web Site in the Content Server

Follow these steps to create a new web site in the content server.

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Expand **Web Content**, and then expand **WEB-INF**.
4. Right-click **wcm-config.xml** and select **Open**.
5. Select the **Server** tab, and then expand **Sites**.
6. Click the **Create** button.
7. In the Create New Site dialog, enter `SiteStudioDemo` for both Name and ID fields.
8. Click **OK**.
9. From the main menu, select **File** and then **Save All**.

4.2.2 Specifying User Credentials for Contribution Mode

Follow these steps to specify user credentials for contribution mode:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Expand **Web Content**, and then expand **WEB-INF**.
4. Right-click **weblogic.xml** and select **Open**.
5. Select **Security** and expand **Security Role Assignments**.
6. Click the **Create** button.
7. For Role Name, enter `WCMContributor`.

8. In the Principals section, click the **Create** button.

   For the newly created security role name add a principal (a user) that matches a principal that exists in your content server. For this tutorial we will use the *weblogic* principal.

9. For Principal Name, enter `weblogic`.
10. From the main menu, select **File** and then **Save All**.

4.2.3 Editing the Web Application Deployment Descriptor

Follow these steps to edit the web application to process JSPF files:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Expand **Web Content**, and then expand **WEB-INF**.
4. Right-click **web.xml** and select **Open**.
5. Select **Pages** and expand **JSP Property Groups**.
6. Click the **Create** button.
7. In the new JSP Property Group, select the **General** tab:
   - For Display Name, enter `Fragments`.
   - For Page Encoding, enter `UTF-8`.
   - Enable the **Is XML** option.
8. Select the **URL Patterns** tab.
9. Click the **Create** button.
10. Enter `*.jspf` as a new URL pattern.
11. From the main menu, select **File**, and then **Save All**.

> **Note:** To set the Environment Encoding to UTF-8, select **Tools** and then **Preferences** from the main menu.

4.2.4 Creating the Home Page

Follow these steps to create a simple home page for your application:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Expand **Site Files**, and then expand **templates**.
4. Right-click **page** and select **New**.
5. In the New Gallery dialog, expand **Web Tier** and select **JSP**.
6. From the Items list, select **JSP**.
7. Click **OK**.
8. In the Create JSP dialog:
   - For File Name, enter `home.jspx`.
   - Enable the **Create as XML Document** option.

- Enable the **Register Site File** option.
- For Asset Type, select **Page Template** from the list.
- For Site File ID, enter `ssd-home-pt`.
- For Description, enter `Homepage for SiteStudioDemo site`.
- Click **OK**.

> **Note:** This tutorial uses JSPX templates stored locally (typically, in a subdirectory within your project on JDeveloper). For additional information on creating and editing JSPX templates see Section 1.10, "Understanding Site Studio JSP/JSPX Templates".

9. After the new page template opens in the editor, select the **Source** tab.
10. This tutorial assumes your default encoding is set to UTF-8,. If you have a different encoding, edit the `encoding` and `charset` entries as needed or replace the entries with this code:

```
<?xml version='1.0' encoding='UTF-8'?>
  <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1">
  <jsp:output omit-xml-declaration="true" doctype-root-element="HTML"
    doctype-system="http://www.w3.org/TR/html4/loose.dtd"
    doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8"/>
```

In the <head> section:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
```

11. Add your style sheet to the <head> section.

> **Note:** You can complete this tutorial without using a style sheet or you can (optionally) add a style sheet to the site at a later time.

For example:

```
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
  <title>SiteStudioDemo</title>
  <link type="text/css" rel="stylesheet"
    href="../wcm/css/template.css"/>
</head>
```

> **Note:** This is an example only. You must provide the correct subdirectory and name of your style sheet. For example, `href="../styles/basicstyle4.css"`.

The SiteStudioDemo site uses a header, menu, sidebar, and footer (JSP Segments). You can use an existing CSS file, create a new style sheet, or copy the sample style sheet provided with this tutorial. See Section 4.5, "Adding a Style Sheet" for more information.

- Code samples in this tutorial reference a style sheet called `template.css`. This is an example only. See Section 4.5.1, "Creating a New Style Sheet" for the steps required to create a new style sheet.
- Code samples in this tutorial reference the Web Content `/wcm/css` subdirectory as the location for this style sheet. This is an example only. See Section 4.5.2, "Creating a Subdirectory for Your CSS File" for the steps required to create this subdirectory.

12. Replace the <body> section with this code:

> **Note:** This code includes references to four JSPF files. These JSP Segments will be created later in this tutorial.

```
<body>
<jsp:include page="/wcm/templates/fragments/header.jspf"/>
  <div id="container">
    <jsp:include page="/wcm/templates/fragments/menu.jspf"/>
    <jsp:include page="/wcm/templates/fragments/sidebar.jspf"/>
    <!-- add content here -->
    <jsp:include page="/wcm/templates/fragments/footer.jspf"/>
  </div>
</body>
```

13. From the main menu, select **File** and then **Save All**.

4.2.5 Associating the Page Template with the Site

Follow these steps to associate the newly created home.jspx page template with your SiteStudioDemo site.

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Expand **Web Content**, then **WEB-INF**, and then **sites**.
4. Right-click **SiteStudioDemo.xml** and select **Open**.
5. In the Site Structure editor, right-click **Home** and choose **Select Primary Page Template**.
6. Select **ssd-home-pt** and click **OK**.
7. From the main menu, select **File** and then **Save All**.

4.2.6 Running the Site and Viewing the Home Page

Followed these steps to view the Home page:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Expand **Web Content**.
4. Right-click **startSiteStudioDemo.jspx** and select **Run**.
5. The integrated WebLogic Server launches and displays startup information in the Log panel. After the application has started, a browser window will open and display the SiteStudioDemo home page.

4.2.7 Creating Site Fragments

Follow these steps to add JSP segments for the header, footer, sidebar, and menu:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Expand **Site Files**.
4. Right-click **Templates** and select **New**.
5. In the New Gallery dialog, select **All Items.**
6. From the Items list, select **Folder (General)**.
7. Click **OK**.
8. In the Create Folder dialog, for Folder Name. enter `fragments`.
9. Click **OK**.

Create a JSP Segment (Header)

10. Right-click the new fragments folder and select **New**.
11. In the New Gallery dialog, expand **Web Tier** and select **JSP**.
12. From the Items list, select **JSP Segment**.
13. Click **OK**.
14. In the Create JSP Segment dialog, for File Name, enter `header.jspf`.
15. Click **OK**.
16. After the new JSP Segment opens in the editor, select the **Source** tab.
17. Add the following code:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1">
 <div id="top">
  <b>
   <a href="#">SiteStudioDemo</a>
  </b>
 </div>
</jsp:root>
```

18. From the main menu, select **File** and then **Save All**.

Create a Second JSP Segment (Footer)

1. Right-click the new fragment folder and select **New**.
2. In the New Gallery dialog, expand **Web Tier** and select **JSP**.
3. From the Items list, select **JSP Segment**.
4. Click **OK**.
5. In the Create JSP Segment dialog, for File Name, enter `footer.jspf`.
6. Click **OK**.
7. After the new JSP Segment opens in the editor, select the **Source** tab.
8. Add the following code:

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1">
 <div id="footer">
  <p align="center">
   Copyright © 2010 | SiteStudioDemo
  </p>
 </div>
</jsp:root>
```

9. From the main menu, select **File** and then **Save All**.

Create a Third JSP Segment (Sidebar)

1. Right-click the new fragment folder and select **New**.
2. In the New Gallery dialog, expand **Web Tier** and select **JSP**.
3. From the Items list, select **JSP Segment**.
4. Click **OK**.
5. In the Create JSP Segment dialog, for File Name, enter `sidebar.jspf`.
6. Click **OK**.
7. After the new JSP Segment opens in the editor, select the **Source** tab.
8. Add the following code:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1">
<div id="sidebar">
  <h1 class="first">The Sidebar</h1>
  <p><!-- add content here -->
  Lorem ipsum dolor sit amet, consectetur adipiscing elit.
  In sit amet lacus sed lacus egestas fringilla eu sit amet leo.
  In hac habitasse platea dictumst.
  </p>
  <h1>Linkroll</h1>
  <ul class="linkroll">
    <li>
      <a href="#">Link 1</a>
    </li>
    <li>
      <a href="#">Link 2</a>
    </li>
    <li>
      <a href="#">Link 3</a>
    </li>
    <li>
      <a href="#">Link 4</a>
    </li>
  </ul>
</div>
</jsp:root>
```

9. Replace the "Lorem ipsum" text with your own content.
10. From the main menu, select **File** and then **Save All**.

Create a Fourth JSP Segment (Menu)

1. Right-click the new fragment folder and select **New**.
2. In the New Gallery dialog, expand **Web Tier** and select **JSP**.

3. From the Items list, select **JSP Segment**.
4. Click **OK**.
5. In the Create JSP Segment dialog, for File Name enter `menu.jspf`.
6. Click **OK**.
7. After the new JSP Segment opens in the editor, select the **Source** tab.
8. Add the following code:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
  xmlns:wcm="http://www.oracle.com/jsp/wcm"
  xmlns:c="http://java.sun.com/jsp/jstl/core"
  xmlns:fn="http://java.sun.com/jsp/jstl/functions">
<div id="menu">
  <ul id="nav">
    <li>
      <wcm:url var="root" type="node"
        url="${wcmContext.project.structure.rootSection.ID}"/>
      <a href="${root}">
        ${wcmContext.project.structure.rootSection.properties.label}
      </a>
    </li>
    <c:set var="sections"
      value="${wcmContext.project.structure.rootSection.activeChildren}"/>

<!-- Iterate through the site structure nodes and add them to the menu -->
    <c:forEach var="section" items="${sections}">
    <wcm:url var="sectionUrl" type="node" url="${section.ID}"/>
    <li>
      <a title="${section.properties.label}" href="${sectionUrl}">
        <em>${fn:substring(section.properties.label, 0, 1)}</em>
        ${fn:substring(section.properties.label, 1, -1)}
      </a>
    </li>
  </c:forEach>
  </ul>
<br class="clear"/>
</div>
</jsp:root>
```

9. From the main menu, select **File** and then **Save All**.

4.2.8 Creating the Element Definitions

Follow these steps to create two element definition files:

1. From the main menu, select **View**, then **Site Studio**, and then **Site Assets**.
2. From the Project drop-down list (top left), select the **SiteStudioDemo** project.

   If your project is not listed, it means you are not logged in to your connection. From the Application Resources panel, expand **Connections**, then **Site Studio**, then right-click the connection and select **Login**.
3. From the Asset Type drop-down list (top right), select **Element Definition**.
4. Click the **Create New Asset Type** button and select **New Plain Text Element Definition**.
5. For both Title and Content ID, enter `ssd-plaintext-ed`.

6. Click **OK**.
7. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.
8. Click the **Create New Asset Type** button again and select **New WYSIWYG Element Definition**.
9. For both Title and Content ID, enter `ssd-wysiwyg-ed`.
10. Click **OK**.
11. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.
12. From the main menu, select **File** and then **Save All**.

4.2.9 Creating a Region Definition

Follow these steps to create a region definition in the content server:

1. From the Asset Type drop-down list (top right), select **Region Definition**.
2. Click the **Create New Asset Type** button and select **New Region Definition**.
3. For both Title and Content ID, enter `ssd-content-rd`.
4. Click **OK**.
5. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.
6. Right-click the **ssd-content_rd** asset and select **Edit**.
7. After the region definition opens in the editor, click **Add**.
8. In the Element Instance dialog:
   - For both Name and Label, enter `title`.
   - For location, click **Browse**.
   - Select the `ssd-plaintext-ed` element definition and click **OK**.
   - Click **OK** again.
9. Click **Add**.
10. In the Element Instance dialog:
    - For both Name and Label, enter `content`.
    - For location, click **Browse**.
    - Select the `ssd-wysiwyg-ed` element definition and click **OK**.
    - Click **OK** again.
11. From the main menu, select **File** and then **Save All**.
12. In the Check In dialog, click **OK**.

4.2.10 Creating a Region Template

Follow these steps to create a region template that uses a region definition file:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).

3. Expand **Site Files** then **Templates**.
4. Right-click **region** and select **New**.
5. In the New Gallery dialog, expand **Web Tier** and select **JSP**.
6. From the Items list, select **JSP**.
7. Click **OK**
8. In the Create JSP dialog:
   - For File Name, enter `content.jspx`.
   - Select the **Create as XML Document** option.
   - Select the **Register Site File** option.
   - Select **Region Template** from the Asset Type drop-down list.
   - For Site File ID, enter `ssd-content-rt`.
   - For Description, enter `Content Region Template`.
   - For Region Definition, click **Browse**.
   - Select the **ssd-content-rd** region definition and click **OK**.
   - Click **OK** again.
9. After the new region template opens in the editor, select the **Source** tab.
10. Add the following code:

```
<?xml version='1.0' encoding='UTF-8'?>
  <jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
    xmlns:wcm="http://www.oracle.com/jsp/wcm">
     <wcm:dataFile var="dataFile"/>
      <div id="content">
        <h1><a href="#">${dataFile.title}</a></h1>
        <p>${dataFile.content}</p>
      </div>
  </jsp:root>
```

11. From the main menu, select **File** and then **Save All**.

4.2.11 Creating a Placeholder Definition

Follow these steps to create a placeholder definition that uses the new region template and new region definition:

1. From the Asset Type drop-down list (top right), select **Placeholder Definition**.
2. Click the **Create New Asset Type** button and select **New Placeholder Definition**.
3. For both Title and Content ID, enter `ssd-content-phd`.
4. Click **OK**.
5. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.
6. Right-click **ssd-content_phd** and select **Edit** to open in the editor
7. Select all the **Allowed Actions** options.
8. For the Region Definition table, click **Add**.
9. Select `ssd-content-rd` and click **OK**.

10. For the Region Template table, click **Add** and enter `ssd-content-rt`.
11. Select `ssd-content-rt` and click **OK**.
12. From the main menu, select **File** and then **Save All**.
13. In the Check In dialog, click **OK**.

4.2.12 Adding a Placeholder to the Home Page

Follow these steps to add a placeholder to home.jspx:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Expand **Site Files**, then **templates**, then **page**.
4. Right-click **home.jspx** and select **Open**.
5. After the home page opens in the editor, select the **Source** tab.
6. From the main menu, select **View**, and then **Component Palette**.
7. On the Component Palette, select **Site Studio** from the drop-down list.
8. In the Insert Placeholder dialog:
   - For Name, enter `content`.
   - Enable the **Add Name to Placeholder Definition Mappings** option.
   - Select **SiteStudioDemo** from the drop-down list.
   - Click **OK**.
9. Your home page should look something like this:

```
<body>
<jsp:include page="/wcm/templates/fragments/header.jspf"/>
  <div id="container">
    <jsp:include page="/wcm/templates/fragments/menu.jspf"/>
    <jsp:include page="/wcm/templates/fragments/sidebar.jspf"/>
    <!-- add content here -->
    <wcm:placeholder name="content"/>
    <jsp:include page="/wcm/templates/fragments/footer.jspf"/>
  </div>
</body>
```

10. From the main menu, select **File** and then **Save All**.
11. Expand your Site Studio project (SiteStudioDemo).
12. Expand **Web Content,** then **WEB-INF**, and then **sites**.
13. Right-click **SiteStudioDemo.xml** and select **Open**.
14. In the Site Structure editor, expand the **Web Site (SiteStudioDemo)** node.
15. Click **Design Mode Role**, and select the **admin** security role from the drop-down list.
16. Click the **Placeholder Definition Mappings** button.
17. From the list, select **content** and click **Edit**.
18. In the Edit Placeholder Definition Mapping dialog:
    - For Primary Definition ID, click **Search**.

- Select **ssd-content-phd** and click **OK**.
- Click **OK** again.

19. Click **OK**.
20. From the main menu, select **File** and then **Save All**.

4.2.13 Assigning Content to the Placeholder

Follow these steps to enter contribution mode and add content to the placeholder on the homepage:

1. Expand your Site Studio project (SiteStudioDemo) and expand **Web Content**.
2. Right-click **startSiteStudioDemo.jspx** and select **Run**.
3. Once the application is loaded, use **Ctrl+Shift+F5** to enter contribution mode.
4. For the **content** placeholder, click the **Assign Content** button.
5. In the Switch Content wizard, enable the **New Contributor Data File** option for the Choose Content File step.
6. Click **Next**.
7. For the Check-in Content step:
   - For Content ID, enter `ssd-home-df`.
   - For Web Sites, make sure that **SiteStudioDemo** is listed (this should be done automatically).
   - For Region Definition, make sure that **ssd-content-rd** is listed (this should be done automatically.).
8. Click **Finish**.
9. Click the **Edit Region Content** button for the content placeholder.
10. On the Elements tab, for Title, enter `Introduction`.
11. In the content editor add some text.
12. Click the **Save and Close** button.
13. Exit contribution mode using **Ctrl+Shift+F5**.

4.3 Creating Sidebar Links

This section builds on the previous sections in this tutorial. Follow these steps to create a static placeholder and display several links on the site sidebar:

- Step 1: Creating a Static List Element Definition
- Step 2: Creating a Sidebar Region Definition
- Step 3: Creating Static List Data File
- Step 4: Creating a Region Template
- Step 5: Updating the Sidebar Fragment
- Step 6: Creating a Sidebar Content Region Definition
- Step 7: Creating a Sidebar Content Region Template
- Step 8: Creating a Sidebar Content Placeholder Definition

- Step 9: Adding a Sidebar Placeholder Mapping to the Site
- Step 10: Assigning Content to the Sidebar

4.3.1 Creating a Static List Element Definition

Follow these steps to create a static list element definition:

1. From the main menu, select **View**, then **Site Studio,** and then **Site Assets**.
2. From the Project drop-down list (top left), select the **SiteStudioDemo** project.
3. From the Asset Type drop-down list (top right), select **Element Definition**.
4. Click the **Create New Asset Type** button and select **New Static List Element Definition**.
5. For both Title and Content ID, enter `ssd-links-staticlist-ed` and click **OK**.
6. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.
7. Right-click the **ssd-links-staticlist-ed** asset and select **Edit**.
8. Click **Elements**.
9. In the Elements dialog, click **Add**.
10. In the Elements Instance dialog:
    - For both Name and Label, enter `label`.
    - For Location, click **Browse**.
    - Select the **ssd-plaintext-ed** element definition and click **OK**.
    - Click **OK** again.
11. Click **Add** and add a second element.
12. In the Elements Instance dialog
    - For both Name and Label, enter `url`.
    - For Location, click **Browse**.
    - Select the **ssd-plaintext-ed** element definition and click **OK**.
    - Click **OK** again.
13. Click **OK** on the Elements dialog.
14. From the main menu, select **File** then **Save All**.
15. In the Check In dialog, click **OK**.

4.3.2 Creating a Sidebar Region Definition

Follow these steps to create a sidebar region definition:

1. From the main menu, select **View**, then **Site Studio,** and then **Site Assets**.
2. From the Asset Type drop-down list (top right), select **Region Definition**.
3. Click the **Create New Asset Type** button and select **New Region Definition**.
4. For both Title and Content ID, enter `ssd-links-rd` and click **OK**.
5. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.

6. Right-click the **ssd-links-rd** asset and select **Edit**.
7. After the region definition opens in the editor, click **Add**
8. In the Element Instance dialog:
   - For both Name and Label, enter `links`.
   - For Location, click **Browse**.
   - Select the **ssd-links-staticlist-ed** element definition and click **OK**.
   - Click **OK** again.
9. From the main menu, select **File** then **Save All**.
10. In the Check In dialog, click **OK**.

4.3.3 Creating Static List Data File

Follow these steps to create a static list data file:

1. From the main menu, select **View**, then **Site Studio,** and then **Site Assets**.
2. From the Asset Type drop-down list, select **Data File**.
3. Click the **Create New Asset Type** button and select **New Contributor Data File**.
4. In the Create New Asset dialog:
   - For both Title and Content ID, enter `ssd-sidebarlinks-df`.
   - For Region Definition, click **Browse**.
   - Select the **ssd-links-rd** region definition and click **OK**.
   - Click **OK** again.
5. From the main menu, select **File** then **Save All**.

4.3.4 Creating a Region Template

Follow these steps to create a region file that uses a region template:

1. Expand **Site Files**, and then expand **templates**.
2. Right-click the **region** node and select **New**.
3. In the New Gallery dialog, expand **Web Tier** and select **JSP**.
4. From the Items list, select **JSP**.
5. Click **OK**.
6. In the Create JSP dialog:
   - For File Name, enter `links.jspx`.
   - Select the **Create as XML Document** option.
   - Select the **Register Site File** option.
   - Select **Region Template** from the Asset Type drop-down list.
   - For Site File ID, enter `ssd-links-rt`.
   - For Description, enter `Sidebar Region Template`.
   - For Region Definition, click **Browse**.
   - Select the **ssd-links-rd** region definition and click **OK**.

- Click **OK** again.

7. After the new region definition opens in the editor, select the **Source** tab.
8. Add the following code:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:wcm="http://www.oracle.com/jsp/wcm"
          xmlns:c="http://java.sun.com/jsp/jstl/core">
  <jsp:directive.page contentType="text/html;charset=UTF-8"/>
  <wcm:dataFile var="dataFile"/>
  <ul class="linkroll">
    <c:forEach var="row" items="${dataFile.links.rows}">
      <li>
        <a href="${row.url}">${row.label}</a>
      </li>
    </c:forEach>
  </ul>
</jsp:root>
```

9. In the above sidebar region template, note how we are using the new ssd-sidebarlinks-df data file and referencing the properties to format the data.
10. From the main menu, select **File** and then **Save All**.

4.3.5 Updating the Sidebar Fragment

Follow these steps to update the sidebar fragment:

1. Expand **Site Files**, then **templates**, and then **fragments**.
2. Right-click the **sidebar.jspf** fragment and select **Open**.
3. After the file opens in the editor, select the **Source** tab.
4. Add the following code:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:wcm="http://www.oracle.com/jsp/wcm"
          xmlns:c="http://java.sun.com/jsp/jstl/core"
          xmlns:fn="http://java.sun.com/jsp/jstl/functions">
  <div id="sidebar">
    <h1 class="first">Sidebar</h1>
    <p>
      <wcm:placeholder name="sidebar"/>
    </p>
    <h1>Linkroll</h1>

    <wcm:staticPlaceholder dataFile="ssd-sidebarlinks-df"
                           template="@ssd-links-rt"/>
  </div>
</jsp:root>
```

5. In the above sidebar region template note how we are using a data file and referencing the properties of the data file.
6. From the main menu, select **File** and then **Save All**.

4.3.6 Creating a Sidebar Content Region Definition

Follow these steps to create a sidebar content region definition:

1. From the Asset Type drop-down list, select **Region Definition**.
2. Click the **Create New Asset Type** button and select **New Region Definition**.
3. For both Title and Content ID, enter `ssd-text-rd` and click **OK**.
4. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.
5. Right-click the **ssd-text-rd** asset and select **Edit**.
6. After the region definition opens in the editor, click **Add**.
7. In the Element Instance dialog:
   - For both Name and Label, enter `text`.
   - For Location, click **Browse**.
   - Select the **ssd-wysiwyg-ed** element definition and click **OK**.
8. Click **OK** again.
9. From the main menu, select **File** and then **Save All**.
10. In the Check In dialog, click **OK**.

4.3.7 Creating a Sidebar Content Region Template

Follow these steps to create a sidebar content region template file:

1. Expand **Site Files**, and then expand **templates**.
2. Right-click **region** and select **New**.
3. In the New Gallery dialog, expand **Web Tier** and select **JSP**.
4. From the Items list, select **JSP**.
5. Click **OK**.
6. In the Create JSP dialog:
   - For File Name, enter `simple.jspx`.
   - Select the **Create as XML Document** option.
   - Select the **Register Site File** option.
   - Select **Region Template** from the Asset Type drop-down list.
   - For Site File ID, enter `ssd-text-rt`.
   - For Description, enter `Text Region Template`.
   - For Region Definition, click **Browse**.
   - Select the **ssd-text-rd** region definition and click **OK**.
7. Click **OK** again.
8. After the file opens in the editor, select the **Source** tab.
9. Add the following code:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
```

```
        xmlns:wcm="http://www.oracle.com/jsp/wcm"
        xmlns:c="http://java.sun.com/jsp/jstl/core">
  <jsp:directive.page contentType="text/html;charset=UTF-8"/>
  <wcm:dataFile var="dataFile"/>
  <p>${dataFile.text}</p>
</jsp:root>
```

10. From the main menu, select **File** and then **Save All**.
11. In the Check In dialog, click **OK**.

4.3.8 Creating a Sidebar Content Placeholder Definition

Follow these steps to create a sidebar content placeholder definition:

1. From the Asset Type drop-down list, select **Placeholder Definition**.
2. Click the **Create New Asset Type** button and select **New Placeholder Definition**.
3. For both Title and Content ID, enter `ssd-simple-phd` and click **OK**.
4. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.
5. Right-click the **ssd-simple-phd** asset and select **Edit**.
6. Select all the **Allowed Actions** options.
7. For the Region Definition table, click **Add**.
8. Select `ssd-text-rd` and click **OK**.
9. For the Region Template table, click **Add**.
10. Select `ssd-simple-rt` and click **OK**.
11. From the main menu, select **File** and then **Save All**.
12. In the Check In dialog, click **OK**.

4.3.9 Adding a Sidebar Placeholder Mapping to the Site

Follow these steps to add a sidebar placeholder mapping to the site:

1. Expand **Web Content**, then **WEB_INF**, and then **sites**.
2. Right-click **SiteStudioDemo.xml** and select **Open**.
3. Click the **Placeholder Definition Mappings** button.
4. In the Placeholder Definition Mappings dialog, click **Add**.
5. In the Define Placeholder Definition Mapping dialog:
   - For Name, enter `sidebar`.
   - For Primary Definition ID, click **Search**.
   - Select `ssd-simple-phd` and click **OK**.
   - Click **OK** again.
6. In the Placeholder Definition Mappings dialog, click **OK**.
7. From the main menu, select **File** and then **Save All**.

4.3.10 Assigning Content to the Sidebar

Follow these steps to enter contribution mode and add content to the placeholders on the sidebar:

1. Expand your Site Studio project (SiteStudioDemo) and expand **Web Content**.
2. Right-click **startSiteStudioDemo.jspx** and select **Run**.
3. Once the application is loaded, use **Ctrl+Shift+F5** to enter contribution mode.
4. Click the **Switch to Design Mode** button (upper right corner).
5. For the sidebar placeholder, click the **Assign Content** button.
6. In the Switch Content wizard, select **ssd-text-rd** for the Choose Content Region step.
7. For the Choose Content File step, select the **New Contributor Data File** option.
8. Click **Next**
9. For the Check-in Content step:
   - For Content ID, enter `ssd-sidebarcontent-df`.
   - For Web Sites, make sure that **SiteStudioDemo** is listed (this should be done automatically).
   - For Region Definition, make sure that **ssd-text-rd** is listed (this should be done automatically.).
10. Click **Finish**.
11. Click the **Switch to Design Mode** button (upper right corner).
12. For the sidebar placeholder, click the **Edit Region Content** button.
13. In the text editor enter `Introduction`.
14. Click the **Save and Close** button.
15. For the ssd-sidebarlinks-df placeholder, click the **Edit Region Content** button.
16. In the links editor, click **Add Row** and then **Edit Row** and add the following links:

    Link 1:
    - For Label, enter `Google`.
    - For URL, enter `http://www.google.com`.
    - Click **Update Row and Close**.

    Link 2:
    - For Label, enter `Yahoo!`.
    - For URL, enter `http://www.yahoo.com`.
    - Click **Update Row and Close**.

    Link 3:
    - For Label, enter `Bing`.
    - For URL, enter `http://www.bing.com`.
    - Click **Update Row and Close**.
17. Click the **Save and Close** button.
18. Exit contribution mode using **Ctrl+Shift+F5**.

4.4 Creating A Dynamic Conversion Page

This section builds on the previous sections in this tutorial. These steps add a dynamic conversion page that demonstrates the use of native documents in the application. Before proceeding, make sure the DynamicConverter component for the content server is enabled and configured properly.

- Step 1: Creating a Conversions Definition
- Step 2: Creating a Page Template for the Native Document
- Step 3: Creating a New Section for the Native Document
- Step 4: Creating a Region Definition for the Native Document
- Step 5: Creating a Region Template for the Native Document
- Step 6: Creating a Placeholder Definition for the Native Document
- Step 7: Adding the Placeholder to the Native Document Page
- Step 8: Assigning Content to the Placeholder on the Native Document Page

4.4.1 Creating a Conversions Definition

Follow these steps to create a conversions definition:

1. From the Asset Type drop-down list, select **Conversions Definition**.
2. Click the **Create New Asset Type** button and select **New Conversions Definition**.
3. For both Title and Content ID, enter `ssd-default-cd` and click **OK**.
4. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.
5. Right-click the **ssd-default-cd** asset and select **Edit**.
6. After the conversion definition opens in the editor, click **Add** to add a dynamic conversion rule to the definition.
7. In the Native Document Conversion Settings dialog:
   - For Name, enter `simple`.
   - For Options, enable **Use simple inline dynamic conversion**
   - Click **OK**.
8. From the main menu, select **File** and then **Save All**.
9. In the Check In dialog, click **OK**.
10. Expand **Web Content**, then **WEB_INF**, and then **sites**.
11. Right-click **SiteStudioDemo.xml** and select **Open**.
12. Expand the **Web Site (SiteStudioDemo)** node.
13. Double-click the **Conversions Definition** value to display the Select a Conversions Definition Dialog. Select the **ssd-default-cd** conversion definition.
14. Click **OK**.
15. From the main menu, select **File** and then **Save All**.

4.4.2 Creating a Page Template for the Native Document

Follow these steps to create a page template for the native document:

1. Expand your Site Studio project (SiteStudioDemo).
2. Expand **Site Files**, and then expand **templates**.
3. Right-click the page node and select **New**.
4. In the New Gallery dialog, expand **Web Tier** and select **JSP**.
5. From the Items list, select **JSP**.
6. Click **OK**.
7. In the Create JSP dialog:
   - For File Name, enter `native.jspx`.
   - Enable the **Create as XML Document** option.
   - Enable the **Register Site File** option.
   - For Asset Type, select **Page Template** from the list.
   - For Site File ID, enter `ssd-native-pt`.
   - For Description, enter `Native Documents Page for the SiteStudioDemo site`.
   - Click **OK**.
8. Add the following content and save the file:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:wcm="http://www.oracle.com/jsp/wcm">
  <jsp:output omit-xml-declaration="true" doctype-root-element="HTML"
              doctype-system="http://www.w3.org/TR/html4/loose.dtd"
              doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"/>
  <jsp:directive.page contentType="text/html;charset=UTF-8"/>
  <html>
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
      <title>SiteStudioDemo - Demo Blog</title>
      <link type="text/css" rel="stylesheet"
            href="../wcm/css/template.css"/>
    </head>
    <body>
      <jsp:include page="/wcm/templates/fragments/header.jspf"/>
      <div id="container">
        <jsp:include page="/wcm/templates/fragments/menu.jspf"/>

        <jsp:include page="/wcm/templates/fragments/sidebar.jspf"/>
        <!--
          Add content below.
         -->
         <jsp:include page="/wcm/templates/fragments/footer.jspf"/>
      </div>
    </body>
  </html>
</jsp:root>
```

4.4.3 Creating a New Section for the Native Document

Follow these steps to associate the new page template with your site:

1. Expand **Web Content**, then **WEB_INF**, and then **sites**.
2. Right-click SiteStudioDemo.xml and select **Open**.
3. Expand the **Web Site (SiteStudioDemo)** node.
4. Right-click Home and select **Add New Section**.
5. In the Add New Section dialog:
   - For Label, enter `About`.
   - For URL, enter `About`.
   - Click **OK**.
6. Right-click on the About section node and select **Include Section in Navigation**.
7. Right-click on the About section node and select **Select Primary Page Template**.
8. In the Select a Page Template dialog, select the **ssd-native-pt** page template.
9. Click **OK**.
10. From the main menu, select **File** and then **Save All**.

4.4.4 Creating a Region Definition for the Native Document

Follow these steps to create a region definition:

1. From the Asset Type drop-down list, select **Region Definition**.
2. Click the **Create New** button and select **New Region Definition**.
3. For both Title and Content ID, enter `ssd-native-rd` and click **OK**.
4. Click the **Refresh** button to refresh the list. This may take a few minures as the new asset must be indexed by the content server before displaying in the list.
5. Right-click **ssd-native-rd** and select **Edit**.
6. Select an element from the table and click **Switch Region Content**.
7. In the Region Content Options dialog, enable the **Create new native document** option.
8. For Document Types, click **Select**.
9. Choose any document type from the list and click **OK**.
10. From the main menu, click **File** then **Save All**.

4.4.5 Creating a Region Template for the Native Document

Follow these steps to create a region template file that uses a region definition file:

1. Expand your Site Studio project (SiteStudioDemo).
2. Expand **Site Files**, and then expand **templates**.
3. Right-click the **region** node and select **New**.
4. In the New Gallery dialog, expand **Web Tier** and select **JSP**.
5. In the Items section, select **JSP**.

6. Click **OK**.
7. In the Create JSP dialog:
   - For File Name, enter `conversion.jspx`.
   - Enable the **Create as XML Document** option.
   - Enable the **Register Site File** option.
   - For Asset Type, select **Region Template** from the list.
   - For Site File ID, enter `ssd-conversion-rt`.
   - For Description, enter `Native File Region Template`.
   - For Region Definition, click **Browse** and select the **ssd-native-rd** region definition.
   - Click **OK**.
8. Add the following content and save the file:

```
<?xml version='1.0' encoding='UTF-8'?>
<jsp:root xmlns:jsp="http://java.sun.com/JSP/Page" version="2.1"
          xmlns:wcm="http://www.oracle.com/jsp/wcm">
 <div id="content">
  <wcm:dynamicConversion rule="simple"/>
 </div>
</jsp:root>
```

4.4.6 Creating a Placeholder Definition for the Native Document

Follow these steps to create a placeholder definition that utilizes the newly created region template and region definition:

1. From the Asset Type drop-down list, select **Placeholder Definition**.
2. Click the **Create New Asset Type** button and select **New Placeholder Definition**.
3. For both Title and Content ID, enter `ssd-native-phd`.
4. Right-click the **ssd-content-phd** asset and select **Edit**.
5. For the Region Definition table, click **Add,** and select `ssd-native-rd`.
6. For the Region Template table, click **Add**, and select `ssd-conversion-rt`.
7. Select all the **Allowed Actions** options.
8. From the main menu, select **File** and then **Save All**.
9. Check-in and close the placeholder definition.

4.4.7 Adding the Placeholder to the Native Document Page

Follow these steps to add a placeholder to the native.jspx file:

1. From the Component Palette, select **Site Studio**.
2. Drag the Placeholder tag below the *Add Content Below* comment.
3. In the Insert Placeholder dialog:
   - For Name, enter native.
   - Select the **Add Name to Placeholder Definition Mappings** option.

- Select **SiteStudioDemo** as the site.
- Click **OK**.

4. Expand **Web Content**, then **WEB_INF**, and then **sites**.
5. Right-click **SiteStudioDemo.xml** and select **Open**.
6. Click the **Placeholder Definition Mappings** button.
7. Select the native Placeholder Definition Mapping and click **Edit**.
8. In the Edit Placeholder Definition Mapping dialog:
   - For Primary Definition ID, click **Search**.
   - Select `ssd-simple-phd` and click **OK**.
   - Click **OK** again.
9. From the main menu, select **File** and then **Save All**.

4.4.8 Assigning Content to the Placeholder on the Native Document Page

Follow these steps to enter contribution mode and assign content to the placeholder on the native document page.

1. Expand your Site Studio project (SiteStudioDemo) and expand **Web Content**.
2. Right-click **startSiteStudioDemo.jsp** and select **Run**.
3. On the About page, use **Ctrl+Shift+F5** to enter contribution mode.
4. Click the **Assign Content** button on the native placeholder.
5. In the Switch Content wizard, select the **ssd-native-rd** region definition.
6. Click **Next**.
7. On the Choose Content File step, select the **Existing Local File** option.
8. Click **Next**.
9. For the Check-in Content step:
   - For both Title and Content ID, enter `ssd-about-native-df`.
   - Browse to the primary file, and select it.
   - For Web Sites, make sure that **SiteStudioDemo** is listed (this should be done automatically).
   - For Region Definition, make sure that **ssd-native-rd** is listed (this should be done automatically.).
10. Use **Ctrl+Shift+F5** to exit contribution mode.
11. The content of the document displays on the About page.

4.5 Adding a Style Sheet

This section provides information related to adding a style sheet to the example site and covers these topics:

- Section 4.5.1, "Creating a New Style Sheet"
- Section 4.5.2, "Creating a Subdirectory for Your CSS File"
- Section 4.5.3, "Adding a Style Sheet to a Page Template"
- Section 4.5.4, "Sample CSS Document"

The SiteStudioDemo site uses a header, menu, sidebar, and footer (JSP Segments). You can use an existing CSS file, create a new style sheet, or copy the sample style sheet provided in this section.

- Code samples in this tutorial reference a style sheet called `template.css`. This is an example only. See Section 4.5.1, "Creating a New Style Sheet" for the steps required to create a new style sheet.
- Code samples in this tutorial reference the Web Content `/wcm/css` subdirectory as the location for this style sheet. This is an example only. See Section 4.5.2, "Creating a Subdirectory for Your CSS File" for the steps required to create this subdirectory.

Graphics (optional)

The sample style sheet provided in this section references several JPG and GIF files. See Section 4.5.4, "Sample CSS Document" to view the sample CSS code provided with this tutorial. If you use this style sheet for the SiteStudioDemo site, you can create your own images, edit the references, or remove the references:

```
body.jpg
content.jpg
header.jpg
menu.jpg
menu_a.jpg
menu_a_hoover.jpg
menu_link.jpg
sidebar.jpg
top.jpg
linkroll.gif
linkroll_hoover.gif
```

4.5.1 Creating a New Style Sheet

Follow these steps to create a new CSS file:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Right-click **Web Content** and select **New**.
4. In the New Gallery dialog, select **All Items**.
5. From the Items list, select **CSS File (HTML)**.
6. Click **OK**.
7. In the Cascading Style Sheet dialog:
   - Provide a name for the new style sheet. For example, `template.css`.

     The code samples in this tutorial reference a style sheet called `template.css` located in the Web Content `/wcm/css` sub-folder. Both the style sheet and directory are examples only. See Section 4.5.2, "Creating a Subdirectory for Your CSS File" for the steps required to create this subdirectory.
   - Click **OK**.
8. Expand your Site Studio project (SiteStudioDemo).
9. Expand **Web Content**, and then expand **css**.
10. Right-click **template.css** and select **Open**.

11. After the CSS file opens in the editor, select the **Source** tab.
12. The SiteStudioDemo site uses a header, menu, sidebar, and footer (JSP Segments). You can create a new style sheet by entering your own code or copy the sample style sheet provided with this tutorial. See Section 4.5.4, "Sample CSS Document" to view the sample CSS code provided with this tutorial.
13. From the main menu, select **File** and then **Save All**.

4.5.2 Creating a Subdirectory for Your CSS File

Code samples in this tutorial reference the Web Content `/wcm/css` subdirectory as the location for the style sheet. This is an example only.

> **Note:** To create subfolders, right-click a folder and select **New**, then **All Items**, then **Folder (General)**, and click **OK**.

Follow these steps to re-create the subdirectory used as the location for the CSS file in this tutorial:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Right-click **Web Content** and select **New**.
4. In the New Gallery dialog, select **All Items**.
5. From the Items list, select **Folder (General)**.
6. Click **OK**.
7. In the New Folder dialog:
   - Provide a name for the new folder. For example, `wcm`.
   - Click **OK**.
8. Expand **Web Content**, and then expand **wcm**.
9. Right-click **wcm** and select **New**.
10. In the New Gallery dialog, select **All Items**.
11. From the Items list, select **Folder (General)**.
12. Click **OK**.
13. In the New Folder dialog:
    - Provide a name for the new sub-folder. For example, `css`.
    - Click **OK**.
14. From the main menu, select **File** and then **Save All**.

4.5.3 Adding a Style Sheet to a Page Template

Follow these steps to add a CSS file to a page template:

1. In the Application Navigator, select the **Projects** panel.
2. Expand your Site Studio project (SiteStudioDemo).
3. Expand **Site Files**, then **templates**, and then expand **page**.

4. Right-click **home.jspx** (if adding a CSS to the home page) and click **Open**.
5. From the main menu select **View**, then **Component Palette**.
6. In the Component Palette, select the **CSS** option from the list.
7. Drag a Project CSS File (for example, template.css) to the <head> section.
8. From the main menu, select **File** and then **Save All**.

4.5.4 Sample CSS Document

```
/* CSS Document */

body {
background-color:#fff;
background-image:url(../images/body.jpg);
background-position:top;
background-repeat:repeat-x;
font-family:Tahoma, Verdana, Arial, Helvetica, sans-serif;
font-size:78%;
color:#666;
margin:0;
}

#top {
width:715px;
padding:12px 25px 15px 25px;
background-image:url(../images/top.jpg);
background-position:top;
background-repeat:no-repeat;
color:#fff;
margin:0 auto;
text-align:right;
}

#top a {
padding:0 10px 0 10px;
color:#fff;
border:none;
}

#top a:hover {
text-decoration:none;
}

#top b {
float:left;
font-weight:normal;
}

#top b a {
border-left:1px dotted #ccc;
border-right:none;
border:none;
}

#container {
width:765px;
margin:0 auto;
background-image:url(../images/header.jpg);
```

```
background-position:top;
background-repeat:no-repeat;
padding:30px 0 0 0;
}

#menu {
width:706px;
margin:0 auto;
text-align:left;
background-image:url(../images/menu.jpg);
background-position:top;
background-repeat:repeat-x;
}

#menu #nav {
margin:0;
padding:0;
float:left;
}

#nav li {
margin:0;
padding:0;
list-style:none;
float:left;
}

#nav li a {
display:block;
width:100px;
text-align:center;
margin:0 1px 0 0;
background-image:url(../images/menu_link.jpg);
background-position:left;
background-repeat:no-repeat;
color:#ccc;
font-family:Trebuchet MS, Verdana, Arial, Helvetica, sans-serif;
font-size:14px;
text-transform:uppercase;
font-weight:bold;
text-decoration:none;
padding:55px 0 23px 0;
border:none;
}

#nav li a em {
font-style:normal;
text-decoration:underline;
}

#nav li a.last {
margin:0;
}

#nav li a:hover {
background-image:url(../images/menu_link.jpg);
background-position:right;
color:#fff;
}
```

```
#nav li a.active {
background-image:url(../images/menu_link.jpg);
background-position:right;
color:#fff;
}

.clear {
clear:both;
}

#sidebar {
width:262px;
float:right;
background-image:url(../images/sidebar.jpg);
background-position:top;
background-repeat:no-repeat;
margin:0 29px 0 0 !important;
margin:0 15px 0 0;
padding:7px 20px 20px 20px;
color:#777;
line-height:18px;
font-size:90%;
word-spacing:-1px;
}

#sidebar h1 {
font-family:Century Gothic, Verdana, Arial, Helvetica, sans-serif;
font-size:20px;
color:#F2C21C;
font-weight:normal;
margin:10px 0 10px 0;
}

#sidebar h1.first {
color:#fff;
margin:10px 0 35px 0;
}

#content {
margin:0 331px 0 20px;
background-image:url(../images/content.jpg);
background-position:top right;
background-repeat:no-repeat;
padding:10px 20px 10px 20px;
}

#content {
line-height:20px;
word-spacing:2px;
overflow: auto;
height: 400px;
}

#content h1, p.ID11A, p.IDDF {
font-family:Trebuchet MS, sans-serif;
font-size:24px;
color:#333;
font-weight:normal;
margin:30px 0 35px 0;
border-bottom:2px solid #F4C833;
```

```
padding:0 0 2px 0;
}

p.IDAC{
font-family:Trebuchet MS, sans-serif;
font-size:16px;
color:#333;
font-weight:normal;
margin:20px 0 20px 0;
letter-spacing:-2px;
font-style: italic;
padding:0 0 2px 0;
}

#content h1 a {
color:#333;
text-decoration:none;
border:none;
}

#content h1 a:hover {
color:#666;
}

#content p {
text-indent:15px;
}

#sidebar .linkroll {
margin:10px 0 10px 0;
padding:0;
border-top:1px solid #eee;
}

.linkroll li {
margin:1px;
padding:0 !important;
padding:1px;
list-style:none;
}

.linkroll li a {
padding:2px 2px 2px 18px;
display:block;
margin:0;
color:#777;
text-decoration:none;
border-bottom:1px solid #eee;
background-image:url(../images/document.gif);
background-position:left;
background-repeat:no-repeat;
}

.linkroll li a:hover {
border-bottom:1px solid #ccc;
background-color:#fbfbfb;
color:#444;
}

h2 {
```

```
font:120% Verdana;
color:#333;
border-bottom:1px dotted #ccc;
}

blockquote {
margin:0;
padding:5px 15px 5px 15px;
background:#f6f6f6;
}

a {
color:#333;
border-bottom:1px solid #ccc;
text-decoration:none;
}

a:hover {
border-bottom:1px solid #999;
}

#footer {
padding:10px;
border-top:2px solid #f6f6f6;
text-align:center;
margin:20px 0 0 0;
}
```

5

Deploying Your Site Studio Project to a Remote Application Server

Site Studio projects can be bundled and deployed to remote application servers. Bundled Site Studio projects can be deployed to a remote WebLogic application server or remote WebSphere application server using JDeveloper or deployed to your remote application server manually.

This topic describes the steps for creating a web project WAR archive and application EAR archive and deploying the archives to a remote WebLogic application server or WebSphere application server.

> **Note:** These steps assume you have a Site Studio project that includes Site Studio Technology if deploying to WebSphere application servers and both Site Studio Technology and ADF Library Web Application Support if deploying to WebSphere application servers. See Section 4.1, "Creating a Site Studio Project and Connection" or the topic *Creating Site Studio Projects* in the online help for information on creating a new Site Studio project or to associate required technologies with an existing project.

This section covers the following topics:

- Section 5.1, "Creating an Application Server Connection"
- Section 5.2, "Creating Web Project WAR and Application EAR Deployment Profiles"
- Section 5.3, "Deploying your Project with JDeveloper"
- Section 5.4, "Deploying your Project Manually"
- Section 5.5, "Editing Connection Information in the WAR/EAR Archive"

5.1 Creating an Application Server Connection

This section describes the steps to create an application server connection.

1. On the main menu, select **File** then **New**.
2. On the New Gallery dialog, select **General** then **Connections**.
3. Select **Application Server Connection** and click **OK**.
4. Use the Create Application Server Connection wizard to create a connection to your remote application server.

- On the Name and Type step, provide a Connection Name and select **WebLogic 10.3** or **WebSphere 7.x** for Connection Type.
- On the Authentication step, provide an administration-level user name and password.
- On the Configuration step (for WebLogic), provide the Host Name and Port for your remote WebLogic application server and provide a WebLogic Domain name if your WebLogic server is configured to distinguish non-administrative server nodes by name. The WebLogic server installation defaults with a single backend node named *myserver* and a listen port of 7001, which are also the defaults on this dialog.
- On the Configuration step (for WebSphere), provide a Host Name, SOAP Connector Port, Server Name (default is SSXA_server1), Target Node, and Target Cell. For Wsadmin Script File location, enter the path to the wsadmin.sh file on your local WebSphere application server or click **Browse** and navigate to the file. For example, /WebSphere/AppServer/bin/wsadmin.sh.
- On the Test step, click **Test Connection** to determine in the information specified successfully establishes a connection with the application server. If the test passes, click **Finish**

5.2 Creating Web Project WAR and Application EAR Deployment Profiles

This section describes the steps to create a web project WAR deployment profile and application EAR deployment profile.

To create a web project WAR deployment profile:

1. From the main menu, select **View** then **Application Navigator**.
2. In the Application Navigator, select the **Projects** panel.
3. Right-click your Site Studio project and select **Project Properties**.
4. On the Project Properties dialog:
   - Select **Deployment**.
   - Accept the **Use Project Settings** default option or select **Use Custom Settings** depending on your configuration.
   - Click **New**.
5. On the Create Deployment Profile dialog:
   - For Archive Type, select **WAR File** from the list. This creates a profile for deploying the Java EE web module (WAR) to an application server. The WAR consists of the web components (JSPs and servlets) and the corresponding deployment descriptor.
   - For Name, provide a descriptive name for the deployment profile. This name is used to identify this deployment profile when you deploy the application or project.
   - Click **OK**.
6. On the Edit WAR Deployment Profile Properties dialog:
   - Select **Platform**.
   - For Default Platform, select **WebLogic 10.3** or **WebSphere 7.x**.

- For Target Connection, accept the default entry of <none>. This allows you to select a connection when deploying your project.
- Click **OK**.

7. On the Project Properties dialog:
   - The new WAR archive will be listed in the Deployment Profiles list.
   - Click **OK**.
8. From the main men, click **File** then **Save All**.

To create an application EAR deployment profile:

1. On the main menu, select **Application** then **Application Properties**.
2. On the Application Properties dialog:
   - Select **Deployment**.
   - Accept the **Use Project Settings** default option or select **Use Custom Settings** depending on your configuration.
   - Click **New**.
3. On the Create Deployment Profile dialog:
   - For Archive Type, select **EAR File** from the list. This creates a profile for deploying the Java EE enterprise archive (EAR) file to an application server. The EAR file consists of the application's assembled WAR, EJB JAR, and client JAR files.
   - For Name, enter the name for the deployment profile. This name is used to identify this deployment profile when you deploy the application or project.
   - Click **OK**.
4. On the Edit EAR Deployment Profile Properties dialog:
   - Select **Application Assembly**.
   - For Java EE Modules, enable the check box for the web project archive (WAR file) you previously created.
   - Select **Platform**.
   - For Default Platform, select **WebLogic 10.3** or **WebSphere 7.x**.
   - For Target Connection, accept the default entry of <none>. This allows you to select a connection when deploying your project.
   - Click **OK**.
5. On the Application Properties dialog:
   - The new EAR archive will be listed in the Deployment Profiles list.
   - Click **OK**.
6. From the main men, click **File** then **Save Al**l.

5.3 Deploying your Project with JDeveloper

To deploy your project with JDeveloper, you must first create a web project WAR deployment profile and application EAR deployment profile.

To deploy the archives to your remote server:

1. From the main menu, select **Application** then **Deploy** and then select the deployment profile you previously created.
2. On the Deployment Action step, select **Deploy to Application Server.**
3. Click **Next**.
4. On the Select Server step, select the application server connection you previously created.
5. Click **Next**.
6. Review the deployment summary and click **Finish**.

Your site is now deployed to your remote application server.

The link to access your site will look something like this:
http://192.0.2.200:7001/4235-4235Project-context-root/4235site/index.html

Note: You must have your application server properly configured and the appropriate libraries installed.

5.4 Deploying your Project Manually

To deploy your project manually, you must first create a web project WAR deployment file and application EAR deployment profile.

To output the archives to your desktop:

1. From the main menu, select **Application** then **Deploy** and then select the deployment profile you previously created.
2. On the Deployment Action step, select **Deploy to EAR**.
3. Click **Next**.
4. Review the deployment summary, making note on the output file location.
5. Click **Finish**.
6. Retrieve the EAR file from the output file directory.

Your project is now available to be moved to your remote application server. Manually transfer your application EAR archive (the web project WAR archive is included in the application EAR archive) to your remote WebLogic application server, then use the WebLogic Administration Console to deploy the application EAR archive.

5.5 Editing Connection Information in the WAR/EAR Archive

This topic describes the procedure for editing the connection information of a Site Studio project bundled for deployment or deployed to a remote application server. Connection information can be edited manually using any of these three methods: by editing the connections.xml and wcm-config.xml files contained in the application EAR and web project WAR archives, by overwriting the previous connection information with a new file containing updated connection information, or by editing the deployed project files on the application server.

A Site Studio project bundled for deployment consists of an application EAR archive with an included web project WAR archive. The EAR archive contains the connections.xml file, which provides connection details The WAR archive contains the wcm-config.xml file, which references a specific connection in the connections.xml file. To edit connection information, three connection entries in the connections.xml file must be edited and (optionally) one entry in the wcm-config.xml file:

- The connections.xml file usually contains multiple connection entries. For each of these connection entries, the reference name, connectionURL, and idcServerURL values need editing. These values provide the connection details for each listed connection.
- The wcm-config.xml file references a specific connection in the connections.xml file. The connnectionName entry is the only value that (optionally) needs editing. This is the connection you are using for your project. By changing this value, you can select a different connection from the connections listed in the connections.xml file.

This section covers the following topics:

- Section 5.5.1, "Manually Editing Connection Information"
- Section 5.5.2, "Manually Overwriting Connection Information"
- Section 5.5.3, "Editing Deployed Files on your Application Server"

5.5.1 Manually Editing Connection Information

The connection information for a Site Studio project can be changed by manually editing the connections.xml file in the application EAR archive (and optionally the wcm-config.xml file in the web project WAR archive). This method is commonly used when only a few connections entries need to be edited. If a large number of connection entries need to be edited, consider overwriting the connections.xml file with a new file containing the correct connection entries.

To unbundle the EAR archive:

Locate the Site Studio project (application EAR archive) that you output for your remote application server and unbundle the EAR archive.

The EAR archive has this structure:

```
yourapplication.ear
|- adf
  |- META-INF
    |- adf-config.xml
    |- connections.xml
|- lib
|- META-INF
|- yourwebproject.war
```

To edit the connections.xml file:

1. In the unbundled EAR archive, locate the connections.xml file in the /adf/META-INF/ directory.
2. Open the connections.xml file in a text-only editor. The connections.xml file often contains multiple connection entries. For each of these connection entries, the reference name, connectionURL, and idcServerURL values require editing. These values provide the connection details for each listed connection.

3. For the desired connection, edit the reference name value as desired.

   For example, change:

   ```
   <Reference name="productionserver7"
   className="oracle.stellent.ridc.convenience.adf.mbeans.IdcConnection" xmlns="">
   ```

   to the new value:

   ```
   <Reference name="server4"
   className="oracle.stellent.ridc.convenience.adf.mbeans.IdcConnection" xmlns="">
   ```

4. Edit the connectionURL value for that connection.

   For example, change:

   ```
    <StringRefAddr addrType="oracle.stellent.idc.connectionUrl">
   <Contents>idc://productionserver7.yourcompany.com:4444</Contents>
   </StringRefAddr>
   ```

   to the new value:

   ```
   <StringRefAddr addrType="oracle.stellent.idc.connectionUrl">
   <Contents>idc://server4.yourcompany.com:4444</Contents>
   </StringRefAddr>
   ```

5. Edit the idcServerURL value for that connection.

   For example, change:

   ```
   <StringRefAddr addrType="oracle.stellent.idc.idcServerURL">
   <Contents>http://productionserver7.yourcompany.com:16200/cs/idcplg</Contents>
   </StringRefAddr>
   ```

   to the new value:

   ```
   <StringRefAddr addrType="oracle.stellent.idc.idcServerURL">
   <Contents>http://server4.yourcompany.com:16200/cs/idcplg</Contents>
   </StringRefAddr>
   ```

6. Repeat these steps to edit other connections in the connections.xml file as desired.
7. Save the connections.xml file.

To unbundle the WAR archive:

In the unbundled application EAR archive, locate your web project WAR archive and unbundle the WAR archive.

The WAR archive has this structure:

```
yourwebproject.war
|- wcm
|- WEB-INF
  |- sites
  |- tags
  |- wcm-config.xml
  |- web.xml
  |- weblogic.xml
```

To edit the wcm-config.xml file:

1. In the unbundled WAR archive, locate the wcm-config.xml file in the /WEB-INF/ directory.

2. Open wcm-config.xml in a text-only editor. The connectionName entry is the only value that (optionally) needs editing. This is the connection you are using for your project. By changing this value, you can select a different connection from the connections listed in the connections.xml file
3. Edit the content server connectionName value as desired.

   For example, change:

   ```
   <contentServer pollerInterval="10000" enablePoller="true"
   connectionName="productionserver7" adminUser="sysadmin"/>
   ```

   to the new value:

   ```
   <contentServer pollerInterval="10000" enablePoller="true"
   connectionName="server4" adminUser="sysadmin"/>
   ```

4. Save the wcm-config.xml file.

To rebundle the WAR and EAR archives:

1. In the unbundled application EAR archive, locate your unbundled web project WAR archive.
2. Rebundle the WAR archive.
3. Rebundle the complete EAR archive (including the rebundled WAR archive).
4. Redeploy the Site Studio project.

5.5.2 Manually Overwriting Connection Information

The connection information of a bundled Site Studio project can be changed manually by overwriting the connections.xml file with a new file. This method is commonly used when a large number of connection entries need to be edited. If only a few connection entries need to be edited, consider manually editing the connections.xml file directly. See Section 5.5.1, "Manually Editing Connection Information" for more information.

1. Using JDeveloper, create a new (blank) Site Studio project.
2. Define your connection information.
3. Create a new web project WAR archive and application EAR archive.
4. Output your archives.
5. Extract the connections.xml file from the new project EAR archive.

   The connections.xml file is located in the /adf/META-INF/ directory of the unbundled EAR archive.
6. Copy the new connections.xml fils and overwrite the old XML file in the old EAR archive.
7. Manually transfer your application EAR archive to your remote application server.
8. Use your application server administration console to redeploy the updated EAR archive with the new connection information.

5.5.3 Editing Deployed Files on your Application Server

The connection information can be edited after deployment to your application server by editing the deployed project files on the application server. After deployment, the project files are part of the deployable unit and are no longer bundled in the pre-deployment EAR and WAR archives. To edit connection information, the connections.xml file and wcm-config.xml file in the deployable unit must be located and edited on your application server.

These steps describe how to locate and edit your deployed Site Studio project files on a running WebLogic application server using the WebLogic Administration Console. Steps for locating and editing the connections.xml file and wcm-config.xml file in the deployable unit on other application servers would be similar. Consult your application server administration guide for information on locating unbundled files.

1. Launch Oracle WebLogic Administration Console.
2. From Domain Structure, select **Deployments**.
3. From Summary of Deployments, select the **Control** tab.
4. Click the link of the deployed Site Studio project you want to edit (for example, mySiteStudioProject).
5. From Settings, select the **Overview** tab.
6. The `Path` value displays the path to the source of the deployable unit on the Administration Server.

   Note: If the source path is relative, it is resolved relative to InstallDir/app if InstallDir is not null. Otherwise, it is resolved relative to the domain root.

   For example:

   \JDeveloper\mywork\mySiteStudioProject\
7. Manually edit the connections.xml file and (optionally) the wcm-config.xml file.

   Note: These files are part of the deployable unit. That is, they are no longer bundled in the pre-deployment EAR and WAR archives.

   - The connections.xml file is located in the /.adf/META-INF/ subdirectory.
   - The wcm-config.xml file is located in the /WEB-INF/ subdirectory.
8. Changes take effect after you restart the updated Site Studio project.

6

Site Studio Application Components and Technology

This section covers the following topics:

- Section 6.1, "Site Studio Application Components"
- Section 6.2, "Site Studio Technologies"
- Section 6.3, "Using Site Studio Technologies in Your Integration"

6.1 Site Studio Application Components

A Site Studio application is an EAR file, containing multiple WAR files that deliver Site Studio web sites.

JavaEE EAR File

The examples in this section uses a Site Studio application called *MyApplication* with a web project called *MySites*. The structure of the EAR file is as follows:

```
+-- MyApplication.ear
    +-- MySites.war
    +-- adf/META-INF/
        +-- connections.xml
    +-- META-INF/
        +-- application.xml
        +-- weblogic-application.xml
```

MySites.war: This is the war file that references a single content server and delivers sites from that content server (see "JavaEE WAR File" on page 6-1).

adf/META-INF/connections.xml: The ADF connections file that holds the definition of which content server is being queried at runtime.

META-INF/weblogic-application.xml: The Weblogic-specific application file that holds a reference to the Site Studio (and RIDC) shared EAR libraries.

JavaEE WAR File

Site Studio web sites are delivered via a WAR file. The Site Studio libraries can communicate with a single Oracle WebCenter Content Server instance per WAR file. The structure of the WAR file is as follows:

```
+-- MySites.war
    +-- wcm/templates/
    +-- WEB-INF/
        +-- web.xml
        +-- wcm-config.xml
        +-- weblogic.xml
```

wcm/templates: Directory containing all local templates (JSP/JSPX files) that are registered in the wcm-config.xml file.

WEB-INF/wcm-config.xml: The main Site Studio configuration file, which lists the available sites, connection name, and other configuration settings.

WEB-INF/weblogic.xml: Weblogic-specific file that holds the reference to the shared Site Studio WAR library.

WEB-INF/web.xml: The standard JavaEE configuration file that lists out the servlets, filters and security settings for the application.

6.2 Site Studio Technologies

Site Studio technology uses servlets, filters and tag libraries to deliver Site Studio sites via the deployed web application.

The examples in this section describe the default configuration for a single Site Studio site using the site ID *MySite* and is delivered via a Site Studio application called *MyApplication*:

- Section 6.2.1, "Site Studio Servlets"
- Section 6.2.2, "Site Studio Filters"

6.2.1 Site Studio Servlets

These are the available Site Studio servlets:

- **SiteStudio Proxy Servlet**: Proxies the content from Oracle WebCenter Content Server to your local application; allows the content server to be treated as a local resource within the domain, allows for single sign-on via identity propagation.
- **Initialization Servlet**: Initializes the Site Studio libraries; does not handle any web requests.

6.2.2 Site Studio Filters

These are the available Site Studio filters:

- **Mode Filter**: Handles the determination of the mode of the current request. The mode will either be `Consumption`, `Contribution`, or `Design`.
- **Site Filter**: Delivers Site Studio web pages using the Site Studio URL syntax, based on the associated project file in Oracle WebCenter Content Server.
- **Proxy Filter**: Proxies the content server content to allow for the content server UI to be served from the local web application.

Mode Filter

The mode filter listens for URLs which map to the contribution mode root (`/wcm-contrib/*`) or the design mode root (`/wcm-design/*`). If the first segment of the URL matches either of these URLs, the mode is set for the request and the

resource is then forwarded, without the first segment, to the web application for processing.

Site Filter

The site filter allows for the processing of Site Studio URLs. Site Studio URLs are hierarchical, based on the structure of the a project file. A typical Site Studio URL looks like `/mysite/about/index.html`. For more information, see Section 1.6, "Understanding the Site URL Format."

Site Studio control flow:

1. Site URL via the browser is requested: `/mysite/about/index.html`
2. Determine the current site ID and section.
   - Matches against `urlPath` in the wcm-config.xml file.
   - The site ID is set to `mysite` and site path set to `/about/index.html`
3. Instantiate a new `SiteContext` object and place it in the HttpServletRequest as an attribute named `wcmContext`.
4. Determine the Page Template ID. In this case, page template is primary via the index.html file name.
5. Lookup the template ID in the wcm-config.xml file to find the local JSP/JSPX resource.
6. Invoke `RequestDispatcher` to include and render the associated JSP/JSPX page.

Proxy Filter

The proxy filter proxies the content server content, using the local user ID, to enable the Oracle WebCenter Content Server user interface to be served from the local web application. This allows the web application to be integrated with Site Studio Contributor and the Oracle WebCenter Content Server user interface pages, which reside in the content server domain.

6.3 Using Site Studio Technologies in Your Integration

This section provides notes on the various Site Studio technologies that can used to provide web content management functionality in external applications by integrating with other technologies and frameworks.

Site Studio technologies, including the Site Studio tag library, Site Studio helper methods, and various Site Studio filters and servlets can be integrated with other technologies and frameworks to deliver web content management functionality to an existing application.

Using Links

When using placeholder tags in externally managed pages or applications, the customer is entirely responsible for the validity of any links in the data file or generated by the region template. The conventional Site Studio section and content links are not relevant to an unmanaged site or application.

Using Site Studio Tags

The `Context`, `Datafile`, and `Template` tags are used in conjunction with `Placeholder` tags, and can be used in externally managed applications.

For more information, see Section 7.1, "Site Studio Tag Library."

Using Site Studio Helper Methods

Site Studio helper methods are available for scripting templates.

For more information, see Section 7.2, "Site Studio Helper Methods."

Integrating with Other Technologies or Frameworks

Site Studio tags can be integrated with other technologies, however, when used in an integration all URL management must be handled by the application or application server. Site Studio does not manage the application structure, perform the URL mapping, or the page templating.

Site Studio content can be combined with other components by adding either a static placeholder or dynamic placeholder to a page or application.

- A static placeholder is a placeholder with an explicitly assigned dDocName (content ID). In other words, the static placeholder is associated with the backing data file (and, optionally, region display template) at design time.
- A dynamic placeholder is a placeholder that stores its backing data file dDocName (and, optionally, region template) in a project file. A dynamic placeholder is associated with a backing data file by business users using the switch region content functionality that is part of the contribution process. A context tag must to be included to define where in the project file the switch content action will write the values generated by the switch content wizard.

7

Site Studio Tag Library and Helper Methods

This section provides information on the Site Studio tag library and the Site Studio helper methods. It covers the following topics:

- Section 7.1, "Site Studio Tag Library"
- Section 7.2, "Site Studio Helper Methods"

> **Note:** Site Studio tags and helper methods are subject to change with each release of Site Studio. For information on tag library service caching see Section 1.9, "Understanding Service Caching with the Site Studio Tag Library."

7.1 Site Studio Tag Library

This topic provides information on the Site Studio tag library and provides descriptions and details for each tag.

- Site Studio Tag Descriptions
- Site Studio <wcm:context> Tag
- Site Studio <wcm:dataFile> Tag
- Site Studio <wcm:dynamicConversion> Tag
- Site Studio <wcm:dynamicList> Tag
- Site Studio <wcm:idcParameter> Tag
- "Site Studio <wcm:idcService> Tag" on page 6
- Site Studio <wcm:metadata> Tag
- Site Studio <wcm:placeholder> Tag
- Site Studio <wcm:staticPlaceholder> Tag
- Site Studio <wcm:url> Tag

7.1.1 Site Studio Tag Descriptions

This section lists each Site Studio tag and provides a brief description. See the specific tag for additional details.

Tag	Description
`wcm:context`	The Context tag creates a new SiteContext object and sets it on the current HttpServletRequest. This overrides any existing SiteContext already present. Subsequent calls to any Site Studio API or tag will use this context.
`wcm:dataFile`	The Data File tag loads content from the content server and parses it into a DataFile object. The object is then exposed as a variable to be used by the Expression Language (EL) to access the individual components of the data file.
`wcm:dynamicConversion`	The Dynamic Conversion tag is used to specify the conversion rule to use when creating a dynamic conversion of a native document.
`wcm:dynamicList`	The Dynamic List tag executes the query from the dynamic list element specified that is in the region definition, and makes the results available in the variable `var`.
`wcm:idcParameter`	The Idc Parameter tag is used in combination with the Idc Service tag to execute a IDC service call on the content server. Takes a name/value pair.
`wcm:idcService`	The Idc Service tag is used in combination with the Idc Parameter tag to execute a IDC service call on the content server.
`wcm:metadata`	The Metadata tag performs a DOC_INFO service call for the content item, and makes the resulting DataBinder available containing the document information about the given content item.
`wcm:placeholder`	The Placeholder tag is used to specify where an item will be inserted in a template. This will typically be content, but it can also be navigation, code, subtemplates, etc.
`wcm:staticPlaceholder`	The Static Placeholder tag can be used to add a placeholder with a fixed data file. This is useful for content that does not change, such as headers and footers.
`wcm:url`	The URL tag renders either hierarchical links into the site structure or links to content server content.

7.1.2 Site Studio <wcm:context> Tag

The Context tag creates a new `SiteContext` object and sets it on the current `HttpServletRequest`. This overrides any existing SiteContext already present. Subsequent calls to any Site Studio API or tag will use this context.

If the context is not initialized from the current path (`initFromPath` is false or not specified), then the new SiteContext will have the same site ID and same URL (unless specified) as the existing SiteContext, if available.

Parameters

- `siteID`: The site identifier. Required.
- `url`: The site path. This is a portion of path minus the site ID (for example, `/about/index.html`. Required.

- `initFromPath`: Set to `true` to initialize the context from the current HttpServletRequest path (the url parameter is then ignored).

Example

Initialize a new context to the primary page of the about section for site ID "mysite":

```
<wcm:context siteID="mysite" url="/about/index.html" />
```

Reset the current SiteContext to a new URL, without specifying the site ID:

```
<wcm:context url="/index.html" />
```

7.1.3 Site Studio <wcm:dataFile> Tag

The Data File tag loads content from the content server and parses it into a DataFile object. The object is then exposed as a variable to be used by the Expression Language (EL) to access the individual components of the data file. Typically the wcm:dataFile tag is used on a region template, to load the assigned DataFile for display.

The DataFile object implements the Map interface, allowing access to all members by the element name. There are two types of elements that can be retrieved:

- `TextElement`: The text content.
- `ListElement`: The static list content. The `ListElement` implements the `List` interface, which each member of the list a map of TextElements.

Parameters

- `var`: The variable name of the resulting DataFile object. Required.
- `dataFile`: The content ID of the data file. If not specified, it uses the currently assigned ID for the current placeholder if called from within a wcm:placeholder tag. Optional.

Example

Assume the data file, with a content ID of `AWARDS` has the following shape:

```
<?xml version="1.0" encoding="UTF-8"?>
<root xmlns="http://www.stellent.com/wcm-data/ns/8.0.0" version="8.0.0.0">
  <element name="title">Awards</wcm:element>
  <list name="list">
    <row>
      <element name="name">2007 Outstanding Organization Award</element>
    </row>
    <row>
      <element name="name">2006 Top 50 Company</element>
    </row>
  </list>
</root
```

From a region template, load the assigned data file:

```
<wcm:dataFile var="dataFile" />
```

Alternatively, load the data file by specifying the content ID:

```
<wcm:dataFile var="dataFile" contentID="" />
```

Model the data into some HTML:

```
<h2>${title}</h2>
```

```
<ul>
<c:forEach var="row" items="${dataFile.list}">
  <li>${row.name}</li>
</c:forEach>
</ul>
```

7.1.4 Site Studio <wcm:dynamicConversion> Tag

The Dynamic Conversion tag is used to specify the conversion rule to use when creating a dynamic conversion of a native document.

Parameter

- `rule`: The name of the dynamic conversion rule as defined in the ConversionsDefinition file. Required.

  From **View > Property Inspector > Rule > Edit** you can select a rule for the dynamic conversion tag. If a rule is not selected, the default rule is used.
- `page`: The page number to display. Optional.
- `var`: Variable to assign the output of the dynamic conversion. If not specified, the output of the conversion will be written to the page directly. Optional.
- `dataFile`: The dDocName (content ID) of the item to convert. If not specified, the data file assigned to the current placeholder will be used. Optional.

Example

Convert the current data file using the rule `ruleName`:

```
<wcm:dynamicConversion rule="ruleName" />
```

Convert the data file `MY_WORD_DOC` using the rule `ruleName`:

```
<wcm:dynamiConversion dataFile="MY_WORD_DOC" rule="ruleName" />
```

7.1.5 Site Studio <wcm:dynamicList> Tag

The Dynamic List tag executes the query from the dynamic list element specified that is in the region definition, and makes the results available in the variable `var`.

Parameters

- `element`: Name of the element within the list used to construct the list element. Required.
- `var`: A dynamic list variable. Required.
- `dataFile`: The data file assigned to the placeholder rendering this dynamic list. Optional.
- `placeholder`: The name of the placeholder rendering this dynamic list. Optional.

Example

Query using the assigned data file in a region template:

```
<wcm:dynamicList element="example" var="example"/>
```

Model the results:

```
<ul>
<c:forEach var="row" items="${example.resultSets.SearchResults.rows}">
  <li>${row.dDocName} - ${row.dDocTitle}</li>
```

```
</c:forEach>
</ul>
```

7.1.6 Site Studio <wcm:idcParameter> Tag

The Idc Parameter tag is used in combination with the Idc Service tag to execute a IDC Service call on the content server. Takes a name/value pair.

Important: Parameters will vary depending on the service called. Refer to the *Oracle WebCenter Content Services Reference Guide* for the list of service calls and parameters.

Parameters

- `name`: The IDC Parameter. Enter the parameter for the IDC Service call made on the content server. Use the Site Studio IdcService Tag to define the IDC Service call to be executed on the content server. Required.
- `value`: An Idc Parameter variable. Enter a valid value for this parameter. Required.
- `file`: If `true`, the value parameter specifies a path (either a local file path or a relative file path in this web application) and will be added to the DataBinder as a file object. Optional.

Example

This example used both the Idc Service tag and Idc Parameter tag. In this example, the CHECKIN_UNIVERSAL service is called and the parameters for the service are defined using name/value pairs.

- The `doFileCopy` parameter is set to TRUE (1), so the file will not be deleted from hard drive after successful check in.
- The `dDocName` parameter defines the Content ID.
- The `dDocTitle` parameter defines the Title.
- The `dDocType` parameter defines the Type.
- The `dSecurityGroup` parameter defines the Security Group.
- The `dDocAuthor` parameter defines the Author.
- The `primaryFile` parameter defines original name for the file and the absolute path to the location of the file as seen from the server.

```
<wcm:idcservice service="CHECKIN_UNIVERSAL" var="callStatus"
  <wcm:idcparameter name="doFileCopy" value="1"/>
  <wcm:idcparameter name="dDocName" value="RemoteTestCheckin23"/>
  <wcm:idcparameter name="dDocTitle" value="Test1"/>
  <wcm:idcparameter name="dDocType" value="ADACCT"/>
  <wcm:idcparameter name="dSecurityGroup" value="Public"/>
  <wcm:idcparameter name="dDocAuthor" value="sysadmin"/>
  <wcm:idcparameter name="primaryFile" value="C:/inetpub/Scripts/query2.asp"/>
/>
```

This example shows how to get data back from the service call:

```
<wcm:idcService service="DOC_INFO_BY_NAME" var="docinfo">
  <wcm:idcParameter name="dDocName" value="DATAFILE1234" />
  <wcm:idcParameter name="RevisionSelectionMethod" value="LatestReleased" />
</wcm:idcService>
<c:forEach var="row" items="${docinfo.resultSets.DOC_INFO.rows}">
  <p>The content item <em>${row.dDocName}</em> is a
```

```
  <u>${row.xWebsiteObjectType}</u></p>
</c:forEach>
```

7.1.7 Site Studio <wcm:idcService> Tag

The Idc Service tag is used in combination with the Idc Parameter tag to execute a IDC service call on the content server.

Important: Parameters will vary depending on the service called. Refer to the *Oracle WebCenter Content Services Reference Guide* for the list of service calls and parameters.

Parameters

- `service`: The Idc Service. Enter the IDC Service call to be executed on the content server. The service call returns a data binder. Use the Site Studio IdcParameter tag to set parameters for the service call. Required.
- `var`: An Idc Service variable. Enter a name for the variable that will hold the results of the returned response from the content server. Required.
- `type`: If set to `stream`, the response from the content server is an InputStream object. By default, the response is a oracle.stellent.ridc.model.DataBinder. Optional.

Example

This example used both the Idc Service tag and Idc Parameter tag. In this example, the CHECKIN_UNIVERSAL service is called and the parameters for the service are defined using field/value pairs

- The `doFileCopy` parameter is set to TRUE (1), so the file will not be deleted from hard drive after successful check in.
- The `dDocName` parameter defines the Content ID.
- The `dDocTitle` parameter defines the Title.
- The `dDocType` parameter defines the Type.
- The `dSecurityGroup` parameter defines the Security Group.
- The `dDocAuthor` parameter defines the Author.
- The `primaryFile` parameter defines original name for the file and the absolute path to the location of the file as seen from the server.

```
<wcm:idcservice service="CHECKIN_UNIVERSAL" var="callStatus"
  <wcm:idcparameter name="doFileCopy" value="1"/>
  <wcm:idcparameter name="dDocName" value="RemoteTestCheckin23"/>
  <wcm:idcparameter name="dDocTitle" value="Test1"/>
  <wcm:idcparameter name="dDocType" value="ADACCT"/>
  <wcm:idcparameter name="dSecurityGroup" value="Public"/>
  <wcm:idcparameter name="dDocAuthor" value="sysadmin"/>
  <wcm:idcparameter name="primaryFile" value="C:/inetpub/Scripts/query2.asp"/>
/>
```

This example shows how to get data back from the service call:

```
<wcm:idcService service="DOC_INFO_BY_NAME" var="docinfo">
  <wcm:idcParameter name="dDocName" value="DATAFILE1234" />
  <wcm:idcParameter name="RevisionSelectionMethod" value="LatestReleased" />
</wcm:idcService>
<c:forEach var="row" items="${docinfo.resultSets.DOC_INFO.rows}">
  <p>The content item <em>${row.dDocName}</em> is a
```

```
  <u>${row.xWebsiteObjectType}</u></p>
</c:forEach>
```

7.1.8 Site Studio <wcm:metadata> Tag

The Metadata tag performs a DOC_INFO service call for the content item, and makes the resulting DataBinder available containing the document information about the given content item.

Parameters

- `contentID`: The content ID. Required.
- `var`: The metadata variable. Required.

Example

```
<wcm:metadata contentID="DATAFILE1234" var="metadata"/>
<c:forEach var="metadatarow" items="${metadata.resultSets.DOC_INFO.rows}">
  <p>The content item <em>${metadatarow.dDocName}</em> is a
  <u>${metadatarow.xWebsiteObjectType}</u></p>
</c:forEach>
```

You can also use the data file on a region template rather than a fixed ID, for example:

```
<wcm:metadata contentID="${wcmContext.placeholder.dataFile}" var="meta"/>
```

7.1.9 Site Studio <wcm:placeholder> Tag

The Placeholder tag is used to specify where an item will be inserted in a template. This will typically be content, but it can also be navigation, code, subtemplates, etc.

Parameters

- `name`: The name of the placeholder. Required.
- `location`: The location on the template. Optional.
- `definition`: Specifies the mapping to a placeholder definition, which overrides any mappings specified in other ways. Optional.
- `template`: The region template to use to render the Data File. Optional.
- `dataFile`: The dDocName of the data file to assign to this placeholder. Optional.
- `regionDefinition`: The dDocName region definition to use, instead of the xRegionDefinition on the data file. Optional.
- `actions`: The allowed actions of the placeholder definition. Any number of actions can be set. If you use parameters that do not work together (for example, specifying a subtemplate as well as a region definition), then the tag will execute based on the order of parameters listed above. Optional.
  - `Update [E]`: Allows contributor update.
  - `Approve [A]`: Allows workflow approval.
  - `Reject [R]`: Allows workflow rejection.
  - `Document Information [I]`: Allows viewing document information.
  - `Switch Data File [S]`: Allows switching the data file.

- `View Usage Report [U]`: Allows viewing the web usage report.
- `View tracker Report [T]`: Allows viewing the web tracker report.
- `Update Document Information [M]`: Allows updating the content information.
- `Switch Region Template [V]`: Allows switching the region template.
- `Remove Content [N]`: Allows content removal by the contributor.

Example

```
<wcm:placeholder name="yourplaceholdername" actions="E"/>
<wcm:placeholder name="yourplaceholdername" actions="EPRISUTMVN"/>
```

7.1.10 Site Studio <wcm:staticPlaceholder> Tag

The Static Placeholder tag can be used to add a placeholder with a fixed data file. This is useful for content that does not change, such as headers and footers.

Parameters

- `dataFile`: The dDocName (content ID) of the data file to assign to this static placeholder. Required.
- `name`: The name of the static placeholder. Optional.
- `template`: The region template to use to render the data file. Optional.
- `actions`: The allowed actions of the placeholder definition. Any number of actions can be set. If you use parameters that do not work together (for example, specifying a subtemplate as well as a region definition), then the tag will execute based on the order of parameters listed above. Optional.
  - `Update [E]`: Allows contributor update.
  - `Approve [A]`: Allows workflow approval.
  - `Reject [R]`: Allows workflow rejection.
  - `Document Information [I]`: Allows viewing document information.
  - `View Usage Report [U]`: Allows viewing the web usage report.
  - `View tracker Report [T]`: Allows viewing the web tracker report.
  - `Update Document Information [M]`: Allows updating the content information.
  - `Switch Region Template [V]`: Allows switching the region template.

Example

```
<wcm:staticPlaceholder datafile="yourdatafile" actions="E"/>
<wcm:staticPlaceholder datafile="yourdatafile" actions="EPRIUTMV"/>
```

7.1.11 Site Studio <wcm:url> Tag

The URL tag renders either hierarchical links into the site structure or links to content server content.

Parameters

- `var`: The variable name to store the link result.

- type: The type of link. Required.
  - node: Defines a URL for a node.
  - dcresource: Defines a URL for a dynamic conversion resource. Only valid inside a placeholder tag.
  - dcpage: Defines a URL for a dynamic conversion page. Only valid inside a placeholder tag.
  - rendition: Defines a URL for a rendition.
  - resource: Defines a URL for a resource. Used to link to weblayout static resources, such as images.
  - Link: Used to create links to content.
- url: The link parameter (format dependent on the type). Required.
  - node: Format of nodeId|nodePath.
  - dcresource: Format of dcResourcePath.
  - dcpage: Format of dcPageNum.
  - rendition: Format of dDocName/renditionName.
  - resource: Format of dDocName or webLayoutPath.
  - Link: Format of dDocName or (nodeId|nodePath)/dDocName.
- siteID: The site identifier for the link target. This consists of the section ID and the content ID for a site content item. For example, a section ID of 56 and content ID of TEST_ITEM.

Example

Create a link to the /About section (which has a section ID of 20):

```
<wcm:url var="url" type="node" url="/About" />
<wcm:url var="url" type="node" url="20" />
```

Create a link to a piece of content in the content server:

```
<wcm:url var="url" type="resource"
  url="/groups/public/documents/document/news_article.doc" />
```

Url to a rendition of a content item::

```
<!--$wcmUrl("resource","dDocName")-->
<!--$wcmUrl("resource","groups/public/documents/adacct/mydocname.jpg")-->
```

Create a link to a piece of content to display inside a node (optionally specifying the section):

```
<wcm:url var="url" type="link" url="NEWS_ARTICLE" />
<wcm:url var="url" type="link" url="20/NEWS_ARTICLE" />
```

7.2 Site Studio Helper Methods

These Site Studio Helper Methods are available for scripting templates:

- Section 7.2.1, "Site Studio <filterSections> Method"
- Section 7.2.2, "Site Studio <listSectionsForRows> Method"
- Section 7.2.3, "Site Studio <isNodeInNavigationPath> Method"
- Section 7.2.4, "Site Studio <lookupSection> Method"

7.2.1 Site Studio <filterSections> Method

Filter a section list to remove inactive sections and contributor-only sections if not in contribution mode.

```
public static List<SectionNode> filterSections (SiteContext siteContext, List
sections) { }
```

Parameters

- `siteContext`: The site context.
- `sections`: A list of {@link SectionNode} objects.
- `stopLevel`: The level to stop (inclusive).
- `includeHome`: True to include the home section

Returns

- `sectionList`: Return a list rows (represented as a list of sections).

Code

```
public static List<SectionNode> filterSections (SiteContext siteContext, List
sections) {
    if (sections == null) {
      return null;
  }

    List<SectionNode> sectionList = new ArrayList<SectionNode> (sections.size ());
    for (Object sectionObj : sections) {
   SectionNode section = (SectionNode)sectionObj;
   if (!Boolean.parseBoolean (section.getModel ().getActive ())) {
  continue;
        }
   if (Boolean.parseBoolean (section.getModel ().getContributorOnly ())
          && !siteContext.isContributorMode ()) {
          continue;
       }

        //add the section to the list
        sectionList.add (section);
    }

    return sectionList;
}
```

7.2.2 Site Studio <listSectionsForRows> Method

Create a list of sections matching the given parameters. Each row in the list will contain all sections at that level.

```
public static List listSectionsForRows (SiteContext siteContext, int startLevel,
int stopLevel, boolean includeHome) { }
```

Parameters

- `siteContext`: The site context.
- `startLevel`: The level to start (inclusive).
- `stopLevel`: The level to stop (inclusive).
- `includeHome`: True to include the home section.

Returns

- `rows`: Return a list rows (represented as a list of sections).

Code

```
   public static List listSectionsForRows (SiteContext siteContext, int
startLevel, int stopLevel, boolean includeHome) {
        List<List<SectionNode>> rows = new ArrayList<List<SectionNode>> ();
         //get the first level
         if (startLevel < 1) {
             startLevel = 1;
         }
         if (stopLevel < 1) {
             stopLevel = 1;
         }
       Project project = siteContext.getProject ();
        for (int i = startLevel; i <= stopLevel; i++) {
             List<SectionNode> sections = project.getStructure
().getSectionsAtLevel (i);
             if (sections.isEmpty ()) {
                 //no more sections so we can end here
                 break;
             }
             rows.add (sections);
         }
       if (startLevel == 1 && includeHome) {
            List<SectionNode> nodes = null;
             if (rows.size () >= 1) {
                 nodes = rows.get (0);
             } else {
                 nodes = new ArrayList<SectionNode> ();
                 rows.add (nodes);
             }
             nodes.add (0, project.getStructure ().getRootSection ());
        }
         return rows;
    }
```

7.2.3 Site Studio <isNodeInNavigationPath> Method

Look to see if the current node is in the navigation path.

```
public static boolean isNodeInNavigationPath
(SiteContext siteContext, String nodeID, boolean includeHome) { }
```

Parameters

- `siteContext`: The site context.
- `nodeID`: The node ID.
- `includeHome`: True to include the home section in this check, false to exclude

Returns

- `isInNavPath`: Returns true if the node ID is part of the current navigation path.

Code

```
    public static boolean isNodeInNavigationPath (SiteContext siteContext, String
nodeID, boolean includeHome) {
        if (nodeID == null) {
```

```
            return false;
        }

        boolean isInNavPath = false;
        SectionNode section = siteContext.getSection ();
        if (section != null) {
            isInNavPath = (section.getID ().equals (nodeID));
             if (!isInNavPath) {
                for (SectionNode parent : section.getAncestors ()) {
                    if (parent.getParent () == null && !includeHome) {
                        continue;
                    }
                    isInNavPath = (parent.getID ().equals (nodeID));
                    if (isInNavPath) {
                        break;
                    }
                }
            }
         }

         return isInNavPath;
    }
```

7.2.4 Site Studio <lookupSection> Method

Look up a section by path or ID.

```
public static SectionNode lookupSection (SiteContext siteContext, String id) { }
```

Parameters

- `siteContext`: The site context.
- `id`: The section path or ID.

Returns

- `section`: Return the section or null if not found.

Code

```
public static SectionNode lookupSection (SiteContext siteContext, String id) {
         //see if this is a number
         SectionNode section = null;
         boolean isNumber = false;
         try {
           Integer.parseInt (id);
           isNumber = true;
        } catch (NumberFormatException exp) {
            //ignored
         }

         if (isNumber) {
             section = siteContext.getProject ().getStructure ().getSectionByID
(id);
         } else {
             //fix up path
           id = PathHelper.ensureForwardSlashes (id);
            //handle relative
             if (!id.startsWith ("/")) {
                 //get the current section
                 SectionNode current = siteContext.getSection ();
```

```
                if (current != null) {
                    id = current.getUrlPath () + "/" + id;
                }
            }
          section = siteContext.getProject ().getStructure ().getSectionByPath
(id);
        }

         return section;
     }
```

Index

Symbols

A

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

X